AD-A235 683

MENTATION PAGE

Form Approved
OMB No. 0704-0188

REPORT DATE	3. REPORT TYPE AND DATES COVERED
14 Feb 91	Final

4. TITLE AND SUBTITLE
An Algorithm for the Near Optimization of Tactical Force Ratios in the Defense

5. FUNDING NUMBERS
Master's Thesis

6. AUTHOR(S)
Mark E. Tillman, CPT, U.S. Army

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Colorado School of Mines Department of Mathematics and Computer Sciences
Golden, CO 80401

8. PERFORMING ORGANIZATION REPORT NUMBER
T-4002

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQDA, MILPERCEN (DAPC-OPB-D)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
This is a thesis submitted to the Faculty and Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree Master of Science (Mathematics).

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
Contains a method which positions tactical units in the defense. The positioning criterion is one which minimizes the force ratios across the battle area. This method maximizes the differential between potential force ratios that could result should units be assigned to oppose each other. A very good, near optimal solution results in as many iterations as units to be assigned. As a result of using this method, the commander and staff could formulate and possibly choose an option which incorporates minimal force ratios. With this option as a standard, the commander and staff can further identify potentially critical flaws in defensive alignments. This thesis contains a simple method which nearly minimizes the tactical ratio of force. This method could be of particular importance at the Division and Corps level. A computer code which is MS-DOS executable is available to assist in calculations and assigning units.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Defense, Force Alignment, Fractional Program, Tactics, Force Ratios, G3's Estimate, Commander's Estimate, Integer Programming, Division and Corps Planning.	112
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassifed	Unclassified	SAR

T-4002

AN ALGORITHM FOR THE NEAR OPTIMIZATION OF TACTICAL FORCE RATIOS IN THE DEFENSE

by

Mark E. Tillman

T-4002

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematics).

Golden, Colorado

Date 14 Feb 91

Signed: Mark E. Tillman
Mark E. Tillman

Approved: R.E.D. Woolsey
Dr. R.E.D. Woolsey
Thesis Advisor

Golden, Colorado

Date 14 Feb 91

Dr. Ardel J. Boes
Professor and Head
Department of Mathematics
and Computer Sciences

ABSTRACT

This thesis contains a method which positions tactical units in a defensive posture against an enemy force. The positioning criterion used is one which minimizes the ratio of enemy force to friendly force. Across the battle area this method minimizes the sum of all ratios of enemy to friendly forces.

The problem of matching units to oppose each other is one of particular importance at the Division and Corps level. Here, as part of the military planning process, staffs analyze different dispositions of friendly forces for the commander. Currently neither the staff nor the commander formulate an option which best minimizes the ratio of forces. One possible reason to avoid this type of formulation is that the mathematics required for solution is not simple. Ironically, though, this ratio of force weighs heavily on the decision of which option to implement. This thesis offers a simple method that nearly minimizes the ratio of forces. The method maximizes the differential between potential force ratios that could result should units be assigned to oppose each other. A very good solution results in as many iterations as units to be assigned.

As a result of using this method, the commander could choose an option which incorporates the critical decision criterion of force ratio in its formulation. With this course of action as a standard, the commander and his staff can further identify potentially critical flaws in defenses.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

ACKNOWLEDGEMENTS

As with any effort of this magnitude, the end product would not be possible without the help of many people. I would like to thank just a few.

Giving more than just time and knowledge, was my academic advisor, Dr. Robert E. D. Woolsey. Dr. Woolsey gave me direction and wisdom during many hours of consultation both in and out of the classroom. To him I am deeply indebted for having shared this experience.

I would also like to recognize Dr. Ruth Maurer and LTC Bruce Goetz. Each aided me invaluably in the production of this work. Through several months, they cheerfully reviewed my drafts and added constructive comments which added credibility to this document in both academic and military circles. To them I am grateful.

Thanks also goes to my brother, Mike, who, as a professional writer, reviewed my final draft. He certainly deserves many thanks.

Thanks, too, goes to my many friends here at school. Kevin Loy, Joe Huber, Jim Watson, and Doug Hart are just a few. Each gave valuable time to me and this work.

Critical in almost everything I have done in life are my parents. In this endeavor they contributed most notably with a direct contribution to my education. The very computer that this work was completed on was a result of a monetary, interest free, loan from my loving parents. Someday I hope to be rich enough to repay the note. Until then I know this thank you will suffice.

Finally, to my wife, Sallie, I sincerely apologize for letting this academic pursuit interfere with our family life. She has been, as always, unflinchingly supportive in

everything I have done. Thank you for giving totally of your time, patience, and love. I know she will accept our diploma as witness to her sacrifices over the past two years of "mind-boggling" math.

Chapter 1

BACKGROUND AND ANNOTATED BIBLIOGRAPHY

This thesis is centered around two problems. The first problem is the military problem. Simply put, this problem is the determination of which friendly units should be assigned opposite which enemy units so that the ratio of forces is minimized. The second problem is the mathematical one which underlies the military one. The mathematical problem can be formulated using integer programming. Integer programming is sometimes used as a mathematical tool to optimize decision making. As it turns out, integer programming itself cannot be directly applied to the military problem. Instead, *fractional* integer programming is required to properly capture the military problem with a mathematical decision means.

1.1 Military Literature Review

First, some background on the military problem. Interestingly, at the unclassified level, no previously published research is available. In the summer of 1990 this problem was researched for several weeks at the Command and General Staff College's Combat Arms Research Library (CARL) at Fort Leavenworth, Kansas. The pursuit for related research included on-line subject, topic, and key word searches in the following data bases: the Defense Technical Information Center (DTIC) which includes over 2 million Department of Defense (DOD) documents; the National Technical Information Service (NTIS) which includes over 200,000 technical documents; and DIALOG which is mostly an applied science data base. Most references found in this search related to studies of simulation results from JANUS (an interactive simulation model currently used by the

Army) and results of full scale force-on-force battles at the National Training Center (NTC). Statistical in nature, none of these studies related to the problem statement considered here. Additionally, several card catalogs at CARL were examined, which refer to information not available on any national on-line data base. These references included a general book collection of over 116,000 military works; a specialized reference collection of military theses, dissertations, student papers, contract studies, and past research efforts; historical military documents from World War II, Korea, and Vietnam; the Combined Arms Center and Command and General Staff College archive collection which contains documents dated before the Civil War; and Army administrative publications (both current and obsolete). In fact, absolutely no information related to the problem of minimizing the ratio of forces prior to conducting a simulation or full scale battle was found in this search.

The problem was also posed to several senior officers at the Army's think tank of subject matter experts at the Center for Army Tactics. One officer, Lieutenant Colonel Pamperl, a senior tactics instructor at the Command and General Staff College, was able to confirm what was becoming a growing suspicion. He stated that the Army has yet to seriously address the problem of minimizing the ratio of forces prior to war fighting.

Several reasons may be cause for this. First, other, more significant factors, so it is hypothesized, may be more critical to success or failure on the battlefield. Many times, human factors may be more important than the number and disposition of tanks or tubes or artillery on the battlefield. This is certainly valid considering the myriad of factors that influence the outcome of battle. Second, the Army presently has a simple system that works.

The current system is outlined in the Command and General Staff College's text CGSC ST 100-9, The Command Estimate. The doctrine outlined in this text calls for the recommendation of the staff as to which of several proposed dispositions is the one the commander should subsequently war game or possibly war fight. Mathematical optimality may not exist in this set of proposed courses of action because each course of action was derived using experience and other intangibles not related to a systematic method of mathematical foundation. Given the enemy disposition, the proposed courses of action are a best guess on friendly force alignment. The decision criterion of force ratios is a tool only considered after the fact and not during course of action development. The final recommendation, however, is based on analyzing the relative combat power of friendly versus enemy forces. This analysis is performed on Form 86-(F626)-3352 (see Appendix B, Figure B-2). After determining the status and capabilities of units, the staff uses this form to make the calculations for relative combat power and force ratios.

Army doctrine stresses that decisions should not be based on force ratios alone (CGSC ST 100-9 [pg 3-2]). Analyzing relative combat power provides conclusions about friendly and enemy capabilities relative to the operation being planned. The comparison of forces provides the planner with a notion of "what to" (capabilities) and not "how to" (operations) fight. The author based the mathematical problem on this notion of "what to" fight. By so doing the force ratio criterion becomes part of the development process for the course of action.

The mathematical problem was formulated as an integer assignment program. Similar to traditional integer assignment problems, this problem seeks to determine which units the commander could assign against which enemy units. Unlike traditional

assignment problems, the objective in this problem is fractional. The fractions represent the force ratios. The next research effort was aimed at optimizing a fractional integer objective function subject to linear assignment constraints.

1.2 Mathematical Literature Review

Surprisingly, very little research was found in the area of fractional programming. A search for all articles related to fractional programming in The Institute of Management Sciences, OR/MS Index, 1952-1987, was conducted. As many as ten or so articles in various journals were written over the past 25 or so years, yet after reviewing each, none were found related to the mathematical problem proposed. In every case, the fractional programming technique discussed applied to linear variables and not integer ones. Exploration of the School of Mines Library produced several titles relating to discrete optimization methods, applied combinatorics, and integer programming but none discussed fractional integer programming. Finally a search was conducted on an on-line Colorado State Library System at four different universities within Colorado. These were: the University of Colorado at Boulder, the University of Colorado at Denver, Colorado State University, and the University of Denver. Several promising titles were located in the Math and Business Libraries at UC Boulder. Unfortunately, none addressed a method for solving the fractional integer program.

1.3 Summary of Research

In summary, the background research for this thesis was extensive and exhaustive. No previously published research on this problem was discovered within the scientific, mathematical, or military communities.

Currently Army doctrine does not call for a mathematically optimal solution to the force ratio alignment problem (CGSC ST100-9 [pg 3-4]). However, by analyzing relative combat power as part of the course of action development process the staff could provide the commander a feel for near minimal relative strength ratios as a basis and standard for possible force alignment.

Chapter 2

PROBLEM DEFINITION AND SOLUTION TECHNIQUE

2.1 Problem Background and Discussion

Commanders and staff officers on today's battlefield are faced with an extremely complex environment. The ever-changing technologies serve to make every thought, action, decision, and disposition of each unit vital. Indeed, the unit that is led and served by the best commanders and staff has the advantage from the outset. When time permits and before men and equipment are committed to a particular strategy, commanders and staffs war game different options for aligning units.

This war gaming is simply a process of thinking systematically about various courses of action and the ensuing chain of events. Prior to beginning this war gaming process, both the commander and staff each analyze the various proposed courses of action for troop alignment. This analysis is known as the estimate of the situation. The estimate is a critical procedural step in the military decision making process. The estimate is normally performed by the staff officers and the commander of the unit. Each section chooses the course of action that best aligns forces against an expected enemy disposition. The decision tool currently used by the G2 (Division or Corps Intelligence Officer) and the G3 (Division or Corps Operations Officer) relies heavily on the concept of tactical force ratios. The tactical force ratio is determined by first assigning relative combat power indices (a number quantifying combat power relative to a fixed standard) to both friend and foe that the staff and commander feel will oppose each other at the onset of the battle (CGSC ST 100-9 [pg 3-3]). Simple division produces the force ratio. When the calculations are finished the staff draws conclusions about friendly and enemy

capabilities and limitations. The force ratio calculations serve as a discriminating or screening criterion for the staff. Courses of action with unfavorable force ratios are possibly eliminated. Whereas courses of action with strong force ratios are favored and given further consideration. Based, in part, on these ratios the G3 recommends a course of action as the initial array of force to begin the war gaming process.

Historically a defender would need at least a ratio of 1 to 3 over an attacker to expect to defend terrain successfully.

2.2 Assumptions

Several assumptions are necessary before a solution method is established. First, an estimate of enemy force concentrations must be known. This in necessary to quantify enemy potential along each of the avenues of approach. Second, the friendly mission is a defensive one. Offensive operations do not directly apply the force ratio analysis in the detail discussed in this thesis. Third, an equal importance is given to the defense of each enemy avenue of approach. Otherwise, a weighting scheme is necessary. Finally, a correlation exists between force ratio minimization and increased success on the battlefield. Without this association these procedures are not applicable to decision making.

2.3 Limitations

For the purpose of this thesis several limitations exist. First, only unclassified information will be discussed and used. Classified sources are not necessary to develop or validate the methods discussed in this thesis. Second, the solution methods discussed in this thesis are of mathematical make-up. This is not an oversight or an omission of

other, possibly more effective, analysis techniques. The objective of this thesis was to identify mathematical methods and tools that could be used to assist staffs and commanders in solving the problem of where to initially deploy units in the defense. The decision criterion used was one of minimizing the ratio of force while adhering to limited tactical prerequisites which will be discussed in chapter 3. The author did not model human factors which could be extremely relevant to the outcome of any battle. Nor did he attempt to simulate a dynamically changing scenario. This author does not intend for the method discussed in this thesis to replace the process currently used by staffs and commanders. Instead, the product of this thesis can be used in conjunction with current Army staff planning doctrine to produce a better, more informed decision.

2.4 Solution Methods

This thesis will focus on two mathematical methods for solving the military problem. The first method formulates a fractional integer program which will determine an optimal solution to the mathematical objective. Two features of this technique will become apparent during its formulation. First, the procedure expends considerable time and effort. Second, the procedure is beyond the expertise of the typical staff officer who might be tasked to propose an array of forces. The second method discussed in this thesis is simple, quick, and persuasive. Furthermore, this second method will determine a near optimal course of action based on the tactical force ratio criterion in a fraction of the time needed to determine optimality using the first method. Once this near optimal course of action is determined, it could supplement the original courses and be analyzed further for practicality. By so doing, the commander could consider a course of action that includes the best available tactical force ratios as the basis for its formulation.

2.4.1 Fractional Integer Programming Method

The problem of optimizing force alignments will be shown to be a fractional integer programming problem. This program has an objective function that is the sum of variable fractions (see eq. 2.1). In this formulation, the objective function would take the form:

$$\sum_{i=1}^{n} \frac{k_i}{a_i} \tag{2.1}$$

Where the denominators a_i are defined as:

for each i, $\sum_{j=1}^{m} c_j x_{ij} = a_i$

Each term in the objective function represents the fractional ratio of force indices for a particular location on the battle front known as an avenue of approach. The numerators of each of the ratios in the objective function (see eq. 2.1) are constant. Each of these constants represents the sum of the enemy combat power indices for a particular avenue of approach. At feasibility, the denominators of these ratios are all non-zero and represent a linear combination of the friendly combat power indices of units chosen to oppose the enemy at a particular avenue of approach. The decision variables x_{ij} represent the decision to assign a unit against an avenue of approach. These decision variables are found in the denominators of the ratios of the terms in the objective function. Each decision variable is constrained so that it may be used only once.

The constraints on the variables are similar to the traditional assignment or transportation problem. Initially all the decision variables x_{ij} are 0-1. The variable is equal to 1 if it is assigned to a term in the objective function (a particular avenue of approach). The constraint set for this formulation is shown below.

Subject to:

for each i, $\sum_{j=1}^{m} x_{ij} \geq 1$

for each j, $\sum_{i=1}^{n} x_{ij} = 1$

Surprisingly enough, this type of problem is not an easy one to optimize using known integer techniques or linear programs. Figure 2.1 illustrates the magnitude of the solution set size for problems of varying sizes. The combinatoric analysis used to determine each solution set size is given in Appendix A, Table A-1.

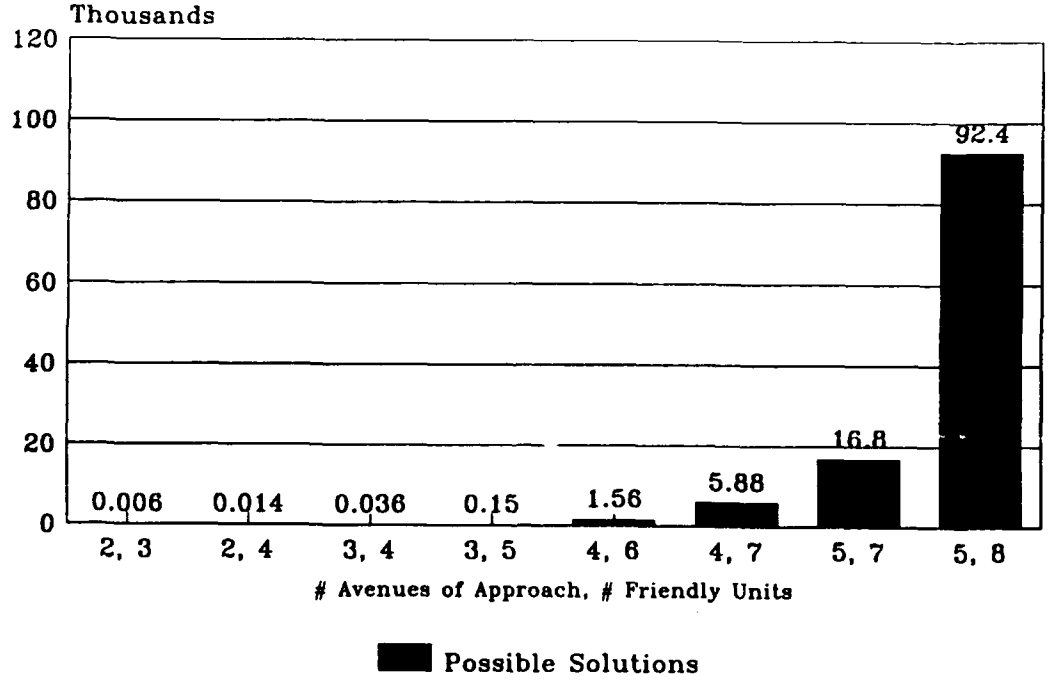


Figure 2.1: Solution Set Sizes

The difficulties in solving such a problem are two-fold. First, the objective function is non-linear. Attempts to linearize the objective function create many new variables and constraints. Second, many of the new variables introduced are not 0-1 like the original variable set. In the end, the program includes not only 0-1 variables, but also integer positive and strictly positive variables. This mixture limits the efficiency of known methods. The time required to formulate and solve the military problem with a fractional integer program may not always be practical. Depicted in Figure 2.2 is the possible solution time for problems of varying sizes. This time does not include time to formulate the problem. The solution times for the problems with four and five avenues of approach were extrapolated from run times for smaller sizes.

Nevertheless, a detailed discussion for this fractional integer formulation and a solution method is discussed in Chapter 3. If time and expertise are not available to address the military problem with a fractional integer programming solution or if an optimal solution is not absolutely necessary, then an approximately optimal result may be preferable.

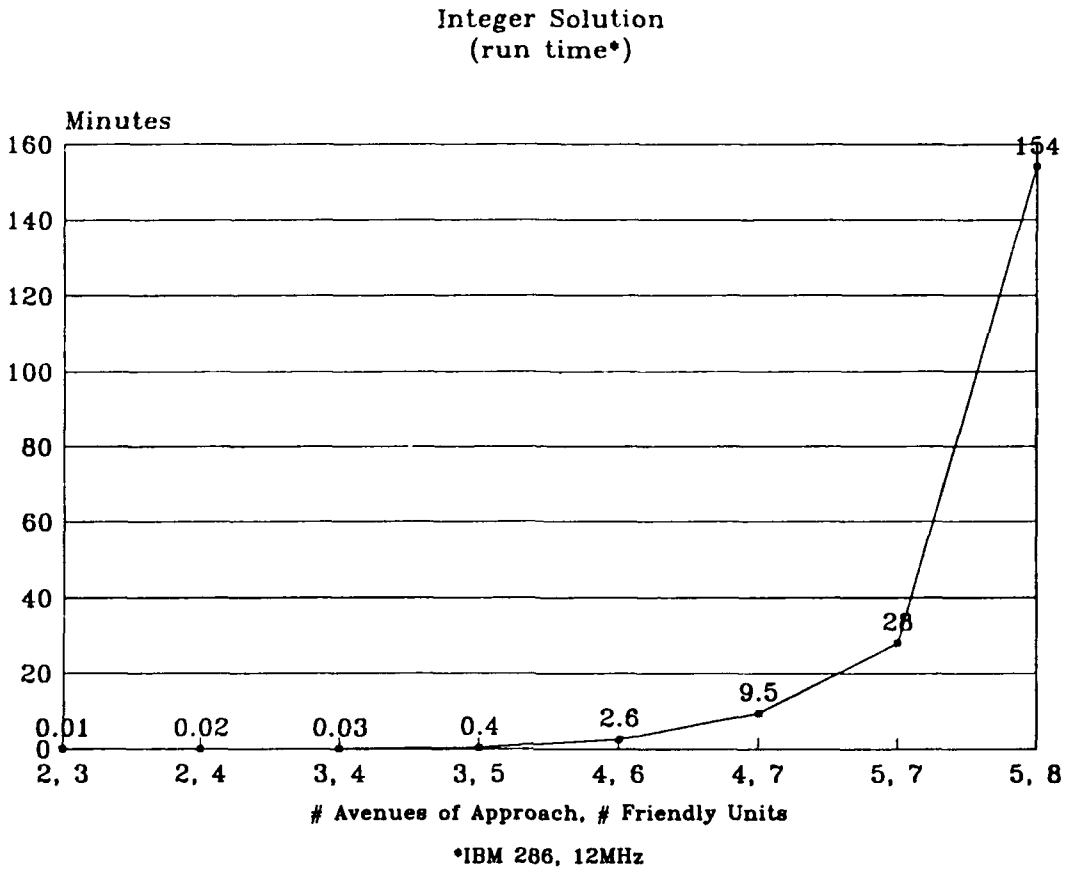


Figure 2.2: Integer Program Computer Run Times

2.4.2 The Near Minimal Algorithm

The algorithm presented in this thesis will determine a near minimum value for the fractional programming problem of the type required to align defensive forces optimally. The algorithm can assign as many as "m" friendly units in no more than "m" iterations. The resulting solution is very close to the minimal value for the fractional integer program. Figure 2.3 reveals the average solution time for problems of varying sizes using the Near-Minimal Algorithm.

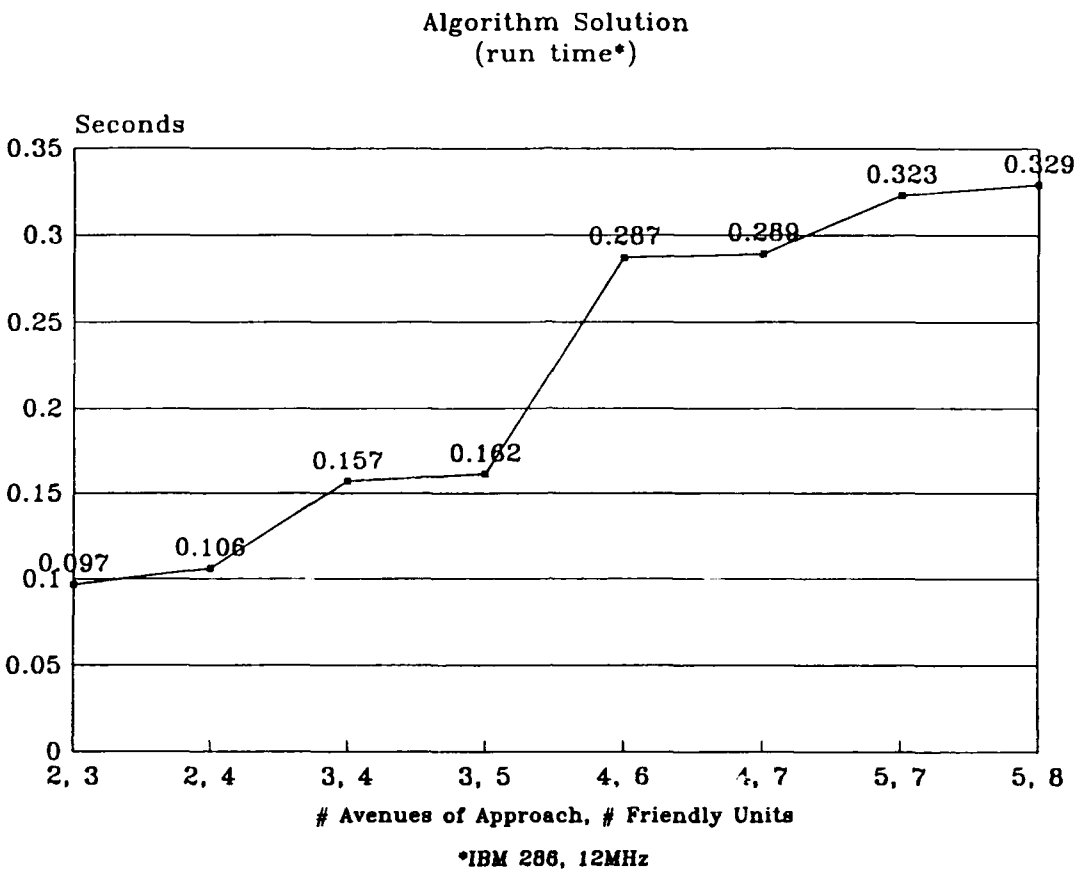


Figure 2.3: Near-Minimal Algorithm Computer Run Times

Note the magnitude of difference in computer run time for each problem when the algorithm run time is compared to the integer program solution time. The near-minimal algorithm produces a very good solution in less than a half of a second. In contrast, the optimal solution may take hours to determine. The time savings is even more apparent when we consider the reality that the problem sizes here are smaller than those expected to be solved by staffs of a Division or Corps. There, the actual size may be eight or more avenues of approach and as many as 20 or so friendly units.

The efficiency of the algorithm is in its execution. Simply stated, the algorithm maximizes the differential between a previous infeasible upper bound solution and a

potential new upper bound solution which is also infeasible until the algorithm's final iteration.

The algorithm proceeds by assigning variables to an appropriate term in the objective function. The choice of which term is made by comparing the marginal difference between the previous upper bound for that term and a new one should the variable be assigned to that term. The greatest marginal difference earns the variable for the term thereby eliminating it from being further assigned to another term. Each iteration moves the objective function non-increasingly toward a newer and lower integer upper bound which is equal to or less than the previous one. At the conclusion of the algorithm, the final integer upper bound is feasible. This final upper bound is very near to the least upper bound for the objective function. At conclusion, the algorithm successfully assigns friendly units to locations where the sum of the resulting force ratios is near-optimal.

2.5 Method Overview

In the pages that follow the reader will find two mathematical solution methods to the same military problem. The first method will optimize the fractional integer objective. This method is discussed in Chapter 3 and illustrated with an example in Chapter 4. The second method will nearly minimize the same fractional integer objective. This method is simpler than the first and is discussed in Chapter 3. Several examples, contained in Chapter 4, will clarify this second method.

Chapter 3

PROBLEM AND SOLUTION METHOD

In this chapter the military problem is presented. Then two mathematical methods are discussed for solving the military problem. The first method uses a fractional integer programming technique and requires extensive reformulation of the original problem. This thesis assumes the reader has a basic understanding of integer programming. The second method is an algorithm which produces a near optimal feasible solution. The algorithm is presented in a manner that assumes the reader is solving the problem by hand using a specially designed worksheet. However, a computer program written in the C language, that implements this algorithm, is shown in Appendix E.

3.1 The Military Problem

Consider a military planning staff faced with the problem of determining where to assign different units to defend against an enemy attack. The organization of each unit varies. Some units may be equipped with modern technology and may be better trained for the upcoming mission. Others may be reserve units which are incompletely manned and poorly equipped. The role of each unit on the battlefield varies, too. Some may be ground gainers such as armor and mechanized infantry. Others may deny access to terrain such as artillery, combat engineers, or aviation assets. The big picture can become very complex, very quickly. The problem is where to assign each unit to maximize friendly combat potential against the enemy. Figure 3.1 illustrates a very simple scenario using current conventional military graphics.

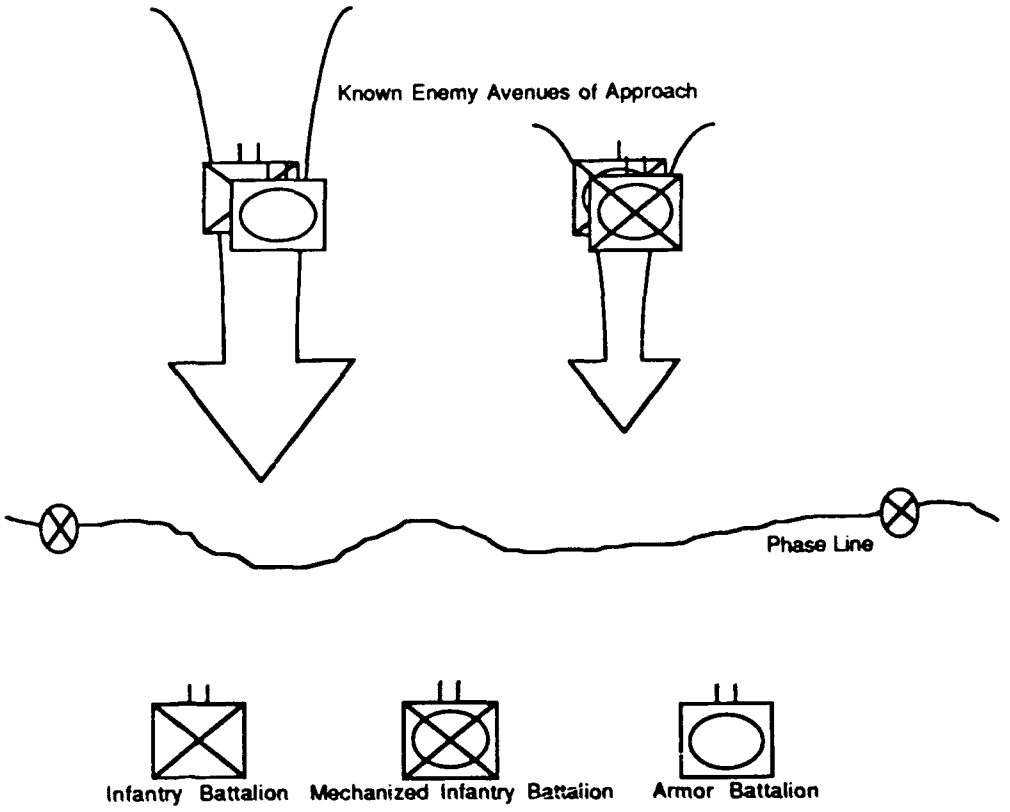
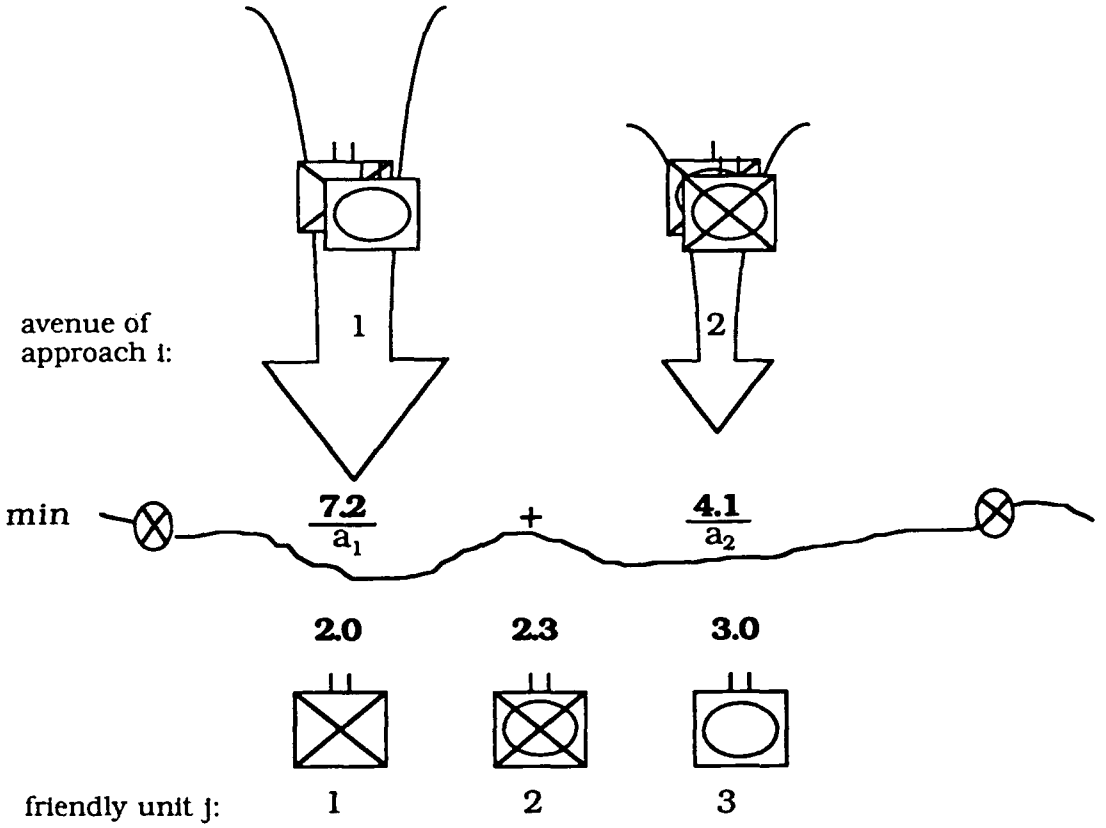


Figure 3.1: Simplified Military Situation

Part of the current decision method in use by the Army is a basic and direct method to capture the different potentials of different units. The Army does this by assigning combat power indices (potentials) to each unit. The indices are relative to a base unit of 1. For example, the more modern equipped American M1 tank battalion may be assigned an index of 3.5. The 3.5 means that this battalion is 3.5 times as lethal as the base unit. Usually the base unit is the Soviet T-55 tank battalion assigned to an independent tank regiment. These indices are arbitrary and subjective but do provide a

simple way of quantifying potential.

With combat power indices now assigned to both friend and foe, the staff must now determine assignments. One important assignment criterion is that of the force ratio. It is believed that the lower the ratio, the better the chance for initial success. For example a ratio of 2 (enemy potential) over 1 (friendly potential) is better than a ratio of 4 to 1. This first example matches relatively greater friendly potential to enemy potential. So the staff analyzes different proposed alignments and determines the force ratios associated with each. It is important to note that none of the proposed alignments were formulated using the force ratio criterion. Instead the staff uses the force ratios to evaluate proposed options. Consequently, the option with the ratio of indices that result in the lowest fractions becomes a favored course of action by the staff. When coupled with other decision criteria such as unity of command and simplicity of execution, this course of action may gain approval by the commander for implementation. The formulation that follows recruits the force ratio criterion as an objective function. The constraints ensure that a unit is assigned in only one place and at least one unit is assigned to oppose every known enemy avenue of approach. If the enemy were unopposed, he could exploit this to his advantage. Therefore it becomes necessary to oppose every known enemy axis of advance (avenue of approach) with at least one friendly unit. This unit could be a radar unit which "listens" for activity or a combat unit which screens or engages and destroys the enemy. Figure 3.2 expands Figure 3.1 to include relevant information for the formulation of the military problem with fractional integer programming. The combat power potentials identified in this figure will be utilized in the example of this method in Chapter 4.



Where: a_i is the sum of friendly indices assigned to avenue i.

Subject to: Every avenue is opposed with a least one friendly unit.

Each friendly unit is assigned only once.

Figure 3.2: Expanded Military Situation

3.2 The Integer Program Formulation

The general form of the fractional integer program can be written as follows:

Minimize:

$$\sum_{i=1}^{n} \frac{k_i}{a_i}$$

Where the denominators a_i are defined as:

for each i, $\sum_{j=1}^{m} c_j x_{ij} = a_i$

Subject to:

for each i, $\sum_{j=1}^{m} x_{ij} \geq 1$ ensuring at least one unit is assigned to each avenue of approach.

for each j, $\sum_{i=1}^{n} x_{ij} = 1$ ensuring that each unit is assigned exactly once.

In this formulation, each term of the objective function represents a different enemy avenue of approach. Each numerator k_i represents the relative combat power of enemy units located within avenue of approach i. The denominators a_i each represent the sum of relative combat power indices of friendly units that are chosen to oppose the enemy units located in avenue of approach i. The force ratio for each avenue of approach, then, is represented by its term in the objective function. Each term should be minimized as much as possible but not at the expense of any other. The effect is that the sum of the force ratios is minimized, which is represented by the objective function.

The decision variable x_{ij} associates the decision to assign or not to assign friendly unit j to avenue of approach i. The constant k_i is the enemy's combat power index associated with avenue of approach i. The constant c_j is the friendly combat power index

associated with friendly unit *j*. The subscript *i* tags the decision variable to a particular avenue of approach. The subscript *j* tags the decision variable to a particular friendly unit.

3.3 The Integer Program Problem Solving Procedure

The steps for solving the fractional integer program are shown in a flow chart (figure 3.3). The integer programming formulation described above is illustrated in example 1 of Chapter 4. The following problem solving technique *appears* to capture the objective of the military problem and uses 0-1 integer programming methodology. It should be noted that one very basic result is not proved in this thesis (see Chapter 5). This procedure relies, instead, on a conjecture. However, acceptance of this conjecture does appear to optimize the decision of which units to assign opposite which avenues so as to minimize the sum of the resulting force ratios.

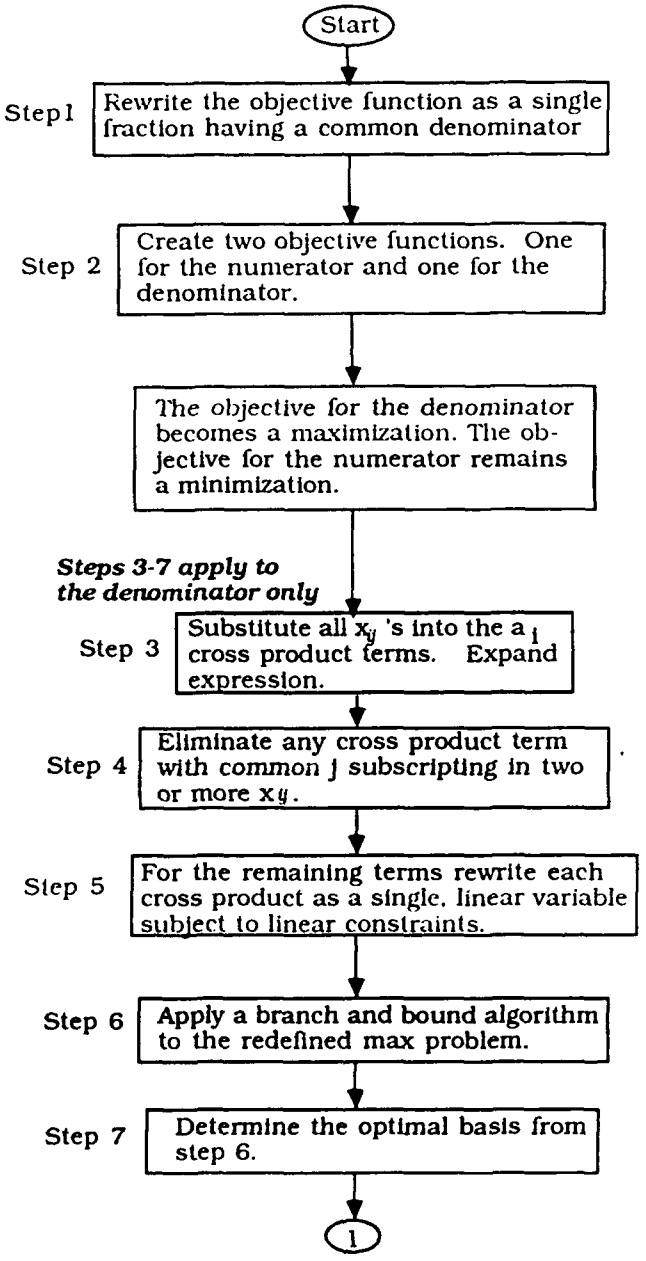


Figure 3.3: Fractional Integer Program Flowchart

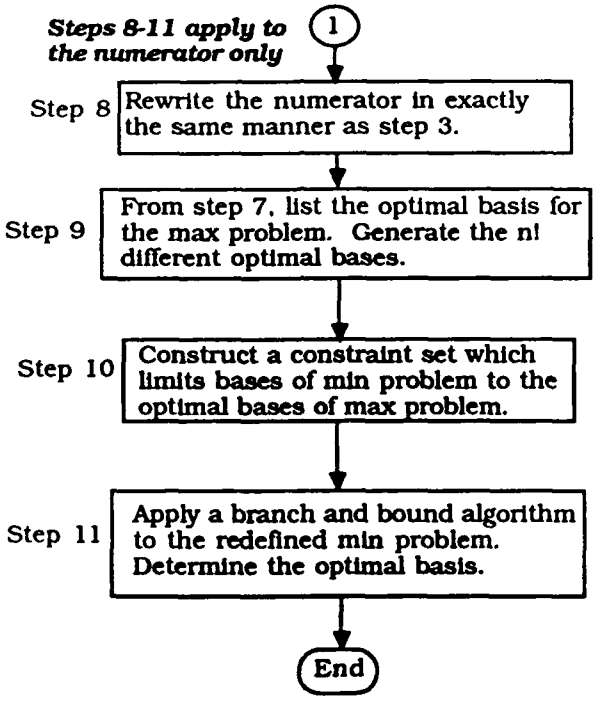


Figure 3.3 (Continued): Fractional Integer Program Flowchart

Suppose the fractional integer objective function is:

$$\sum_{i=1}^{3} \frac{k_i}{a_i}$$

Which expands to:

$$\frac{k_1}{a_1} + \frac{k_2}{a_2} + \frac{k_3}{a_3}$$

STEP 1: Rewrite the objective function over a single common denominator. This gives a new objective function in the form:

$$\frac{k_1 a_2 a_3 + k_2 a_1 a_3 + k_3 a_1 a_2}{a_1 a_2 a_3}$$

Go to step 2.

STEP 2: Create two objective functions, one for the numerator and one for the denominator of the expression from step 1. In the steps that follow, we will first maximize the denominator, then minimize the numerator. In this example, the numerator objective becomes:

Minimize: $k_1 a_2 a_3 + k_2 a_1 a_3 + k_3 a_1 a_2$

Similarly, the denominator objective becomes:

Maximize: $a_1 a_2 a_3$

Go to step 3.

3.3.1 Maximizing the Denominator (Steps 3 thru 7):

STEP 3: Substitute all x_{ij}'s and their coefficients into the a_i cross product expressions and expand the resulting denominator into a polynomial with cross product x_{ij} terms only. For example suppose: $c_1 = 2$, $c_2 = 3$, and $c_3 = 4$. Furthermore, suppose each a_i is defined as:

$a_1 = c_1x_{11} + c_2x_{12} + c_3x_{13}$,

$a_2 = c_1x_{21} + c_2x_{22} + c_3x_{23}$,

Therefore an a_i cross product expression becomes:

$a_1a_2 = (2x_{11} + 3x_{12} + 4x_{13})(2x_{21} + 3x_{22} + 4x_{23})$

Which expands to:

$4x_{11}x_{21} + 6x_{11}x_{22} + 8x_{11}x_{23} + 6x_{12}x_{21} + 9x_{12}x_{22} + 12x_{12}x_{23} + 8x_{13}x_{21} + 12x_{13}x_{22} + 16x_{13}x_{23}$

Go to step 4.

STEP 4: Eliminate any cross product term with common j subscripting in two or more x_{ij}'s within the cross product. The assignment constraints cause these terms to equal zero because friendly unit j can be assigned only once. In this example, the terms containing $x_{11}x_{21}$, $x_{12}x_{22}$, and $x_{13}x_{23}$ would be eliminated. Go to step 5.

STEP 5: For the remaining terms, rewrite each decision variable cross product as a single linear variable with combined subscripting. The fact that the new variable is linear is of great importance because this limits the number of nodes that need to be searched in the branch and bound algorithm. For instance $x_{11}x_{22}$ would become x_{1122}.

Next, introduce linear constraints that will cause the new variable to function like

the original decision variable cross product. For each decision variable cross product introduce the following constraint sets:

For the double product let $x_{ij}x_{i'j'} = x_{iji'j'}$ subject to the constraints:

$$
\begin{array}{lllll}
x_{ij} + x_{i'j'} - x_{iji'j'} & \leq & 1 \\
x_{ij} - x_{iji'j'} & \geq & 0 \\
x_{i'j'} - x_{iji'j'} & \geq & 0
\end{array}
$$

Where the primed subscripting denotes a change in subscript values on the decision variable. Table 3.1 lists the possible values for x_{ij} and $x_{i'j'}$. This table further demonstrates the consistency of values for *the cross product* $x_{ij}x_{i'j'}$ and *the new linear variable* $x_{iji'j'}$ subject to the above constraint set.

Table 3.1: Double Product Truth Table

x_{ij}	$x_{i'j'}$	$x_{ij}x_{i'j'}$	$x_{iji'j'}$
1	1	1	1
1	0	0	0
0	1	0	0
0	0	0	0

For example the linear variable x_{1122} would be substituted for the cross product $x_{11}x_{22}$ and would then be subject to the constraints:

$$x_{11} + x_{22} - x_{1122} \leq 1$$

$$x_{11} - x_{1122} \geq 0$$

$$x_{22} - x_{1122} \geq 0$$

In similar fashion, let the triple product $x_{ij}x_{i'j'}x_{i''j''} = x_{iji'j'i''j''}$ be subject to the constraints:

$$x_{ij} + x_{i'j'} + x_{i''j''} - x_{iji'j'i''j''} \leq 2$$

$$x_{ij} - x_{iji'j'i''j''} \geq 0$$

$$x_{i'j'} - x_{iji'j'i''j''} \geq 0$$

$$x_{i''j''} - x_{iji'j'i''j''} \geq 0$$

Go to step 6.

STEP 6: Apply a branch and bound algorithm to the redefined objective function and constraint set from step 6. Remember only the *original* x_{ij}'s (e.g. x_{11}) are defined as 0-1. The new variables (e.g. x_{1122}) introduced after redefining the cross products are linear and do not need to be defined as 0-1. In this example the redefined objective function found in step 4 is:

Max $6x_{11}x_{22} + 8x_{11}x_{23} + 6x_{12}x_{21} + 12x_{12}x_{23} + 8x_{13}x_{21} + 12x_{13}x_{22}$

Go to step 7.

STEP 7: Determine the optimal basis for the maximization of the denominator. This optimal basis may not be the only basis that attains optimality for this objective. Indeed, a degeneracy of sorts may exist. The branch and bound algorithm will determine one optimum. However, there may be many optima which exist along parallel branches

of the algorithm. It is this possibility of multiple optima (more than one set of decision variables that achieve the same maximum) that leads to the minimization of the numerator after fixing the denominator size at its maximum. Go to step 8.

3.3.2 Minimizing the Numerator (Steps 8 thru 11):

STEP 8: In exactly the same manner as in Step 3, rewrite the numerator of the objective function. Likewise, redefine all cross products and introduce constraint sets in exactly the same manner as step 5. Go to step 9.

STEP 9: From step 7 list the denominator's optimal basis from the branch and bound algorithm. In this step, determine all bases that are optimal for the denominator problem and constrain the numerator problem to include just one of these. This author conjectures that one of these bases is optimal for the original fractional integer program.

First it is important to remember the meaning of the *ij* subscripting on each x_{ij}. The *i* represents the objective function term that the x_{ij} is being assigned to (a particular avenue of approach). Whereas the *j* identifies the friendly unit index. It is possible to attain the same optimal value for the denominator objective with a different feasible combination of x_{ij}s equal to 1. In effect, we want to preserve the grouping of units to a particular avenue of approach. Exactly which avenue will not be known until the numerator is minimized. This can be performed by permuting the *i* (the avenue of approach) subscripting across the *j* subscripting found in the optimal basis in Step 7.

For example, suppose the optimal basis in step 7 is x_{11}, x_{22}, and x_{23}, where each of these variables takes on the value 1. Also suppose $a_1 = c_1x_{11}$ and $a_2 = c_2x_{22} + c_3x_{23}$. Consequently, the cross product, a_1a_2, would then equal $c_1(c_2+c_3)$. Another optimal basis

for the maximization objective could be x_{21}, x_{12} and x_{13}. Where $a_1 = c_1x_{21}$ and $a_2 = c_2x_{12} + c_3x_{13}$. The cross product, a_1a_2, would also equal $c_1(c_2+c_3)$ which is equal to the result of the first basis. In fact, it is possible to find up to $n!$ different optimal bases to the maximization problem, so long as we preserve the grouping of j's (units) in the manner discussed above. Now list the $n!$ different optimal bases to the denominator problem. Go to step 10.

STEP 10: This author conjectures that one of the bases in Step 9 will be optimal for the numerator problem and the original objective function. In this step, construct a constraint set of $n! + 1$ different constraints and introduce $n!$ new variable "switches" y_i that will activate constraints to select the appropriate optimal basis. These new constraints and variable switches will cause one of these sets of variables to be optimal in the numerator problem. In effect, we are now combining the maximization of the denominator and minimization of the numerator into one problem.

Let's revisit the example in Step 9 to illustrate how this is accomplished. We want to choose either the basis x_{11}, x_{22}, and x_{23} or x_{21}, x_{12} and x_{13} during the minimization objective of the numerator. The following constraints will affect this choice:

$$x_{11} + x_{22} + x_{23} - 3y_1 = 0$$

$$x_{12} + x_{13} + x_{21} - 3y_2 = 0$$

$$y_1 + y_2 = 1$$

Both y_1 and y_2 are linear variables and do not need to be defined as 0-1. The fact that y_1 and y_2 are linear is of great importance because this limits the number of nodes that need to be searched in the branch and bound algorithm. Go to step 11.

STEP 11: Apply a branch and bound type algorithm to the redefined minimization objective in step 8 subject to the constraint sets generated in Steps 8 and 10 and the original assignment constraints. I conjecture that the optimal basis from this numerator objective is the optimal basis for the original fractional integer programming problem. This ends the procedure.

3.4 Step-by-Step Procedure for the Near-Minimal Algorithm

The steps for establishing a near-optimal solution to the military problem are shown in a flow chart (figure 3.5). The algorithm that follows is further illustrated in example 2 in Chapter 4. The following procedure captures the objective of the military problem and uses elementary, binary mathematics such as addition and division. The worksheet in Appendix B should be used in conjunction with the procedure. The underlying principle of the algorithm is to maximize the differential between two numbers. The first number represents the current upperbound for an objective term (the current force ratio for avenue i). The second number represents a potentially new upperbound for the same term should a unit be assigned there. In this way we identify the so-called "greatest bang for buck." In so doing, each iteration of the algorithm moves the objective function non-increasingly toward the minimum. Furthermore, each iteration of the algorithm assigns a different friendly unit and corresponds to a different row of Table 1 (see figure 3.4) of the worksheet.

In each row of Table 1 columns are designated for each avenue of approach or term in the objective function. The value of box b represents the current value of the denominator of each term (a_i) in the objective function (See Formulation Section 3.2). The value of box c will contain the potentially new value of the denominator (a_i) should

the unit listed in that row be assigned there. The value of box d will contain the current force ratio for each term in the objective. The value in box e will contain the potential force ratio should the unit listed in that row be assigned there. Finally, the value of box f represents the *marginal contribution* for the unit listed in that row toward minimizing each of the objective function terms. When circled, the entry in box f will represent the *greatest marginal contribution* toward minimizing the objective. This procedure identifies the column (avenue of approach) with the greatest entry in box f for each iteration and assigns the unit listed in that row to this column (avenue of approach). Table 2 of the worksheet records the assignments and aids in calculating the resulting force ratios. Feasibility is guaranteed through the selective incrementing of a counter in column u of Table 1. The value in box m represents the maximum feasible number of units that can be co-assigned to all avenues of approach. Any co-assignment of units to the same avenue of approach increments u. Each iteration of the algorithm compares u to m. When u becomes equal to or greater than m, then this indicates that each successive iteration ***must*** assign remaining units to avenues without a unit initially assigned. Otherwise, an avenue may be unopposed which violates tactical feasibility.

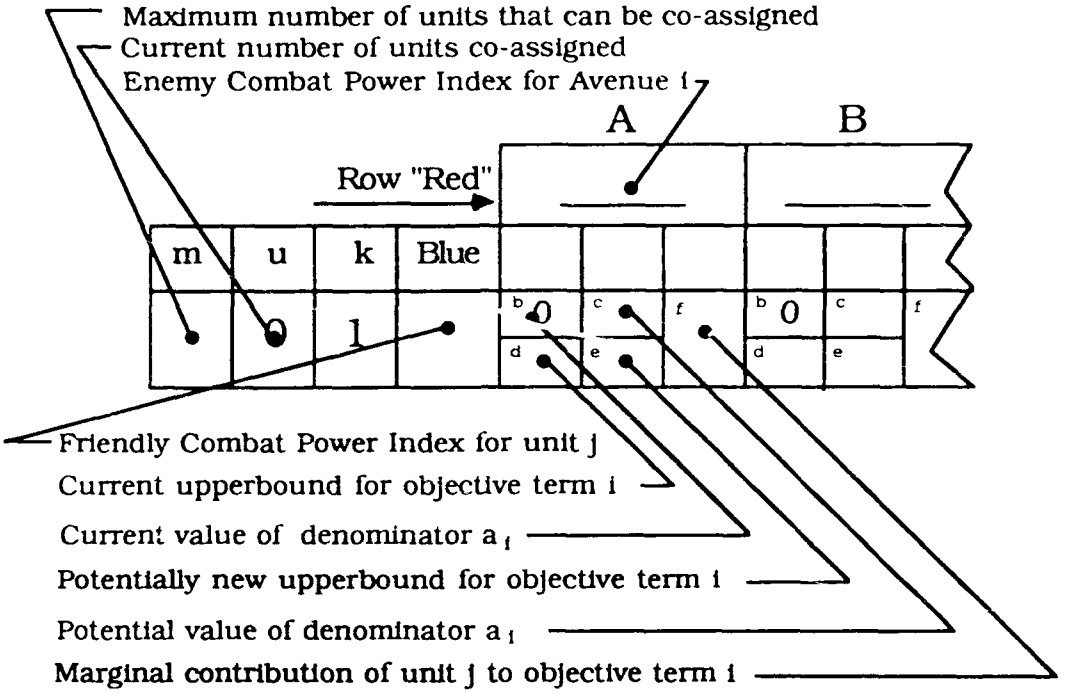


Figure 3.4: Exploded Worksheet Row

At the conclusion of this procedure, all friendly units are assigned according to Army Doctrine (a feasible solution to the military problem) and the sum of the resulting force ratios is very near the minimal value for the fractional integer program discussed earlier.

For larger problems it may be helpful to use the analogous computer program shown in Appendix E. This program is merely an automated worksheet. A tremendous time savings is the major advantage for using the program. Problems of very large size can be solved in less than half a second using the program. The manual method is possibly prone to analyst error and may take 20 or more minutes to complete for larger problems.

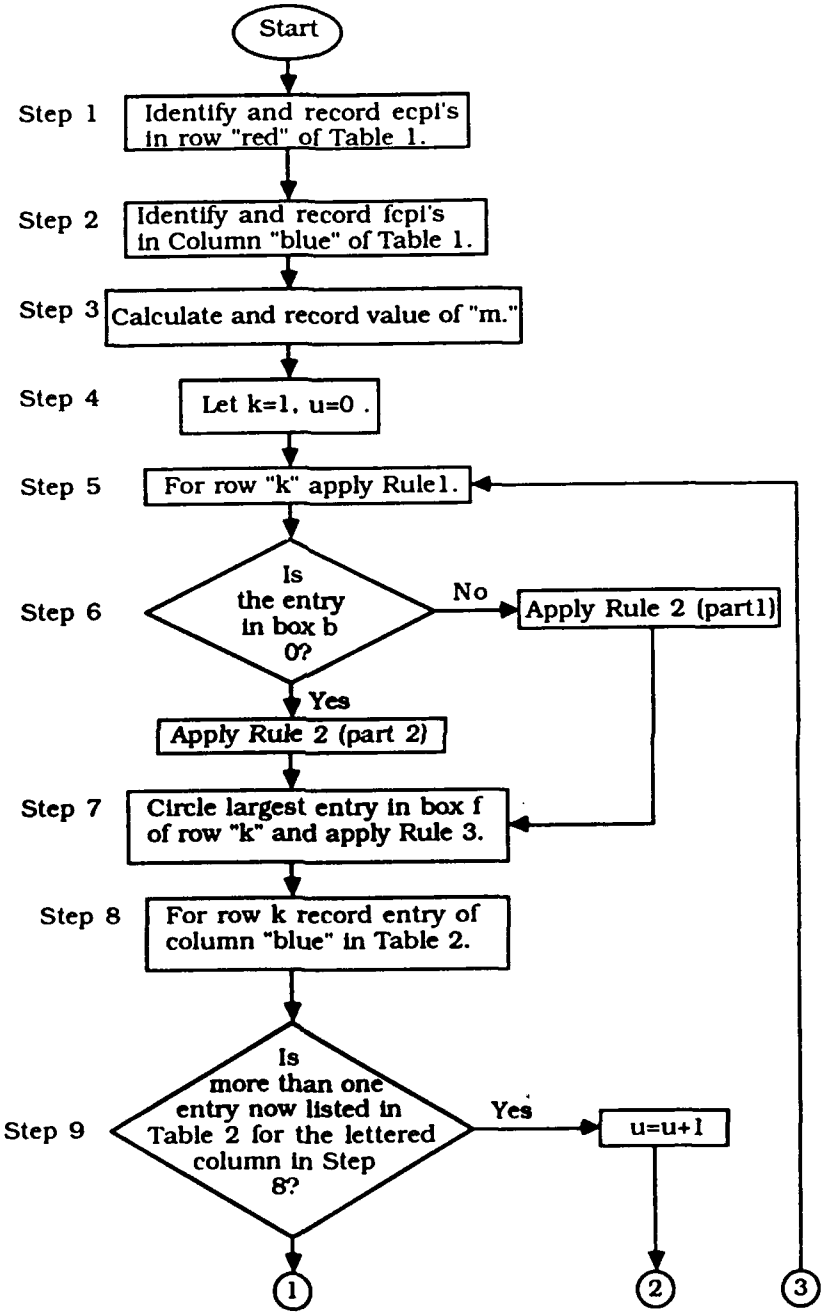
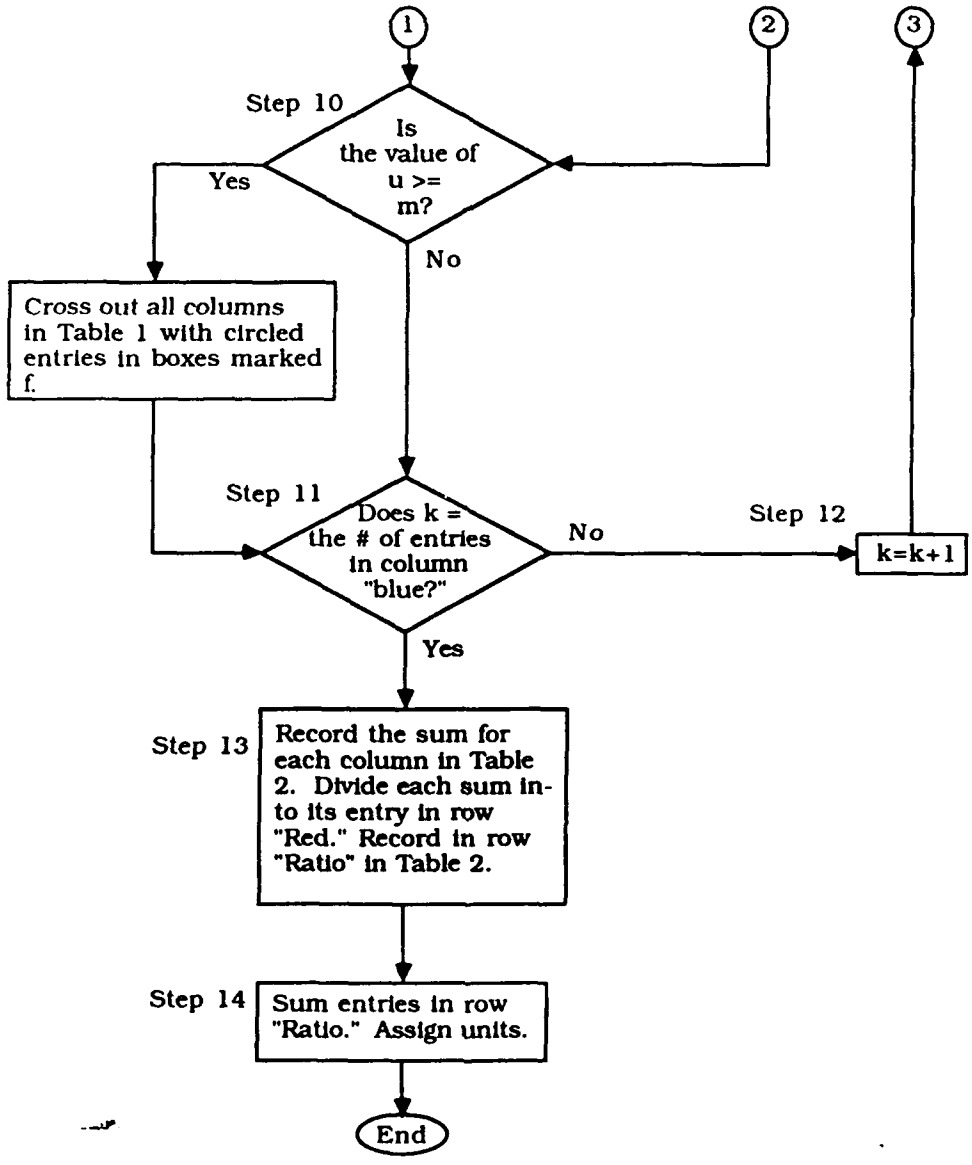


Figure 3.5: Near-Minimal Algorithm Flowchart



Notes:

Ecpi is an abbreviation for Enemy Combat Power Index.
Fcpi is an abbreviation for Friendly Combat Power Index.

Figure 3.5 (Continued): Near-Minimal Algorithm Flowchart

STEP 1: Identify enemy avenues of approach. For each avenue of approach sum the enemy combat power indices* (*ecpi*) in the enemy's first echelon and record each in a separate column of row "red" of Table 1. Cross out all lettered columns without entries in row "red." Enter the total number of avenues of approach in the "Total # Entries" box to the right of row "red." Go to Step 2.

*Refer to page 3-3, CGSC ST 100-9, to obtain unclassified combat power indices for different unit types.

STEP 2: Identify friendly units available for deployment. Assign each their friendly combat power index* *(fcpi)*. Arrange in non-increasing (decreasing) order. Record them in order in column "Blue" of Table 1. Record the number of entries in column "Blue" in the "Total # Entries" box at the bottom of Table 1. Go to Step 3.

*Refer to page 3-3, CGSC ST 100-9, to obtain unclassified combat power indices for different unit types.

STEP 3: Subtract the value in the "Total # Entries" box to the right of row "red" from the "Total # Entries" box at bottom of column "Blue." Record in the box marked m at upper left of Table 1. Go to Step 4.

STEP 4: Let a Table 1 row counter "k" be set equal to 1. Let u=0. (Note that both counters have columns labeled in Table 1 for recording their values throughout this procedure.) Go to Step 5.

STEP 5: For row k of Table 1, apply Rule 1.

RULE 1:

For each used column in row k:

Add the value of column "Blue" to the value in box b.

Record the result in box c.

Go to Step 6.

STEP 6: For row k in Table 1 apply Rule 2. Go to Step **7.**

RULE 2:

For all used columns, divide the value in box c into the value in row "red".

Record in box e.

If the used column has a 0 in box b, *then*:

Subtract the value in box e from the value in row "red."

Record result in box f.

Else, divide the value in box b into the value in row "red."

Record in box d.

Subtract the value in box e from the value in box d.

Record in box f.

STEP 7: For the current row (row k), circle the largest value of all boxes marked f. Apply Rule 3.

RULE 3:

For the column with the circled entry only, add the value in column "Blue" of row k to the value in box b.

Record in box b in next row (row k+1).

For all other capital lettered columns, copy the value in box b to box b of the next row (row k+1).

Go to Step 8.

STEP 8: For the current row (row k), record the value in column "Blue" to the same column of Table 2 as the entry circled in Step 7. Go to Step 9.

STEP 9: If more than one entry is now listed for the capital lettered column in Table 2 from Step 8, then $u=u+1$. Otherwise u is unchanged. Record this new u in column u of row k+1 of Table 1. Go to Step 10.

STEP 10: Is the value of $u \geq m$? If so, cross out all capital lettered columns in Table 1 with circled entries in boxes marked f. No further calculations are necessary for these columns. Go to Step 11.

STEP 11: Is the number of rows filled in equal to the number of entries in column "Blue?" If no, go to Step 12. If yes, go to Step 13.

STEP 12: Let k=k+1 and go to Step 5.

STEP 13: For each capital lettered column of Table 2 sum its entries and record in row "total" in Table 2. Divide this sum into the value in row "Red" of Table 1. Record the result in row "Ratio" of Table 2. Go to Step 14.

STEP 14: Sum all entries in row "Ratio" of Table 2. This sum is the near minimal value for the force ratio problem. Assign units with the combat power indices listed in Table 2 to the Avenue of Approach corresponding to the lettered column they are listed. This ends the procedure.

Chapter 4

EXAMPLE PROBLEMS

This chapter illustrates both the fractional integer programming procedure and the near-minimal algorithm with several examples. The first example is a fractional integer program of meager size. However, notice the length and complexity of the problem solving. All successive examples illuminate the near-minimal algorithm.

4.1 Example 1 (Using Fractional Integer Programming)

4.1.1 Problem Statement

Minimize the force ratios between 3 friendly units with combat power indices of 3, 2.3, and 2 and enemy units assigned along 2 avenues of approach with combat power indices of 7.2 and 4.1. Figures 3.1 and 3.2 graphically illustrate this first example.

4.1.2 Fractional Integer Program Formulation

The fractional integer formulation was discussed in detail in Chapter 3. This method will serve to choose the units for each avenue of approach to minimize the expression below which, in effect, minimizes the sum of ratios of the enemy's combat power indices to the units chosen to oppose it there. The fractional programming problem which formulates the example above would look like the following:

Minimize:

$$\frac{7.2}{a_1}+\frac{4.1}{a_2}$$

Let $x_{ij} = 1$, if along avenue of approach i, friendly unit j is assigned. Otherwise $x_{ij} = 0$.

Where:

$$a_1 = 3x_{11} + 2.3x_{12} + 2x_{13}$$

$$a_2 = 3x_{21} + 2.3x_{22} + 2x_{23}$$

subject to:

$$x_{11} + x_{12} + x_{13} \geq 1$$

$$x_{21} + x_{22} + x_{23} \geq 1$$

$$x_{11} + x_{21} = 1$$

$$x_{12} + x_{22} = 1$$

$$x_{13} + x_{23} = 1$$

4.1.3 Integer Programming Problem Solving Procedure:

STEP 1: By combining the objective function terms into a single fraction, the objective function becomes:

$$\textbf{min } \frac{7.2a_2 + 4.1a_1}{a_1 a_2} \qquad (4.1)$$

STEPS 2-3: From chapter 3, it was conjectured that the minimum of eq. 4.1 will exist at a maximum of a_1a_2. Expanding this expression yields:

Max $a_1a_2 =$

$$9x_{11}x_{21} + 6.9x_{12}x_{21} + 6x_{13}x_{21} + 6.9x_{11}x_{22} + 5.29x_{12}x_{22} + 4.6x_{13}x_{22} + 6x_{11}x_{23} + 4.6x_{12}x_{23} + 4x_{13}x_{23}$$

STEP 4: Since all x_{ij}'s are either 0 or 1 and subject to the assignment constraints above, some terms can be eliminated which will equal 0 at feasibility. The Maximization problem now becomes:

Max $a_1a_2 = 6.9x_{12}x_{21} + 6x_{13}x_{21} + 6.9x_{11}x_{22} + 4.6x_{13}x_{22} + 6x_{11}x_{23} + 4.6x_{12}x_{23}$

STEP 5: Next it will be necessary to redefine each cross product in the objective function as a new variable subject to additional linear constraints.

Let $x_{ij}x_{i'j'} = x_{iji'j'}$

Subject to the additional constraints:

$x_{ij} + x_{i'j'} - x_{iji'j'} \leq 1$

$x_{ij} \quad - x_{iji'j'} \geq 0$

$x_{i'j'} - x_{iji'j'} \geq 0$

For this problem the above constraint types are formulated below:

x_{12}	$+x_{21}$	-	x_{1221}					<=1
x_{12}		-	x_{1221}					>=0
	x_{21}	-	x_{1221}					>=0
x_{13}	$+x_{21}$		-	x_{1321}				<=1
x_{13}			-	x_{1321}				>=0
	x_{21}		-	x_{1321}				>=0
x_{11}		$+x_{22}$		-	x_{1122}			<=1

$$
\begin{array}{lllllll}
x_{11} & & & - & x_{1122} & & >=0 \\
 & +x_{22} & & - & x_{1122} & & >=0 \\
x_{13} & +x_{22} & & & - \; x_{1322} & & <=1 \\
x_{13} & & & & - \; x_{1322} & & >=0 \\
 & x_{22} & & & - \; x_{1322} & & >=0 \\
x_{11} & & +x_{23} & & - \; x_{1123} & & <=1 \\
x_{11} & & & & - \; x_{1123} & & >=0 \\
 & & x_{23} & & - \; x_{1123} & & >=0 \\
x_{12} & & +x_{23} & & & - \; x_{1223} & <=1 \\
x_{12} & & & & & - \; x_{1223} & >=0 \\
 & & x_{23} & & & - \; x_{1223} & >=0
\end{array}
$$

STEP 6: Apply a branch and bound algorithm to this objective function and constraints.

STEP 7: The optimal basis is: $\{x_{11}, x_{22}, x_{23}\}$.

STEP 8: Examining the numerator as the minimization problem we have:

Min $7.2a_2 + 4.1a_1$,

Expanding this expression after substituting the appropriate x_{ij}'s for a_1 and a_2 we have:

Min $21.6x_{21} + 16.56x_{22} + 14.4x_{23} + 12.3x_{11} + 9.43x_{12} + 8.2x_{13}$

STEP 9: We can generate $n!$ different solution sets which yield the same value for the objective function. In this example n (section 3.2), the number of avenues of approach, is 2. This degeneracy of sorts is caused by the x_{ij} sums in the expressions a_1

and a_2. Subject to the above constraints, each of these expressions can actually take on n different combinations of these 0-1 variables in which the coefficients of the variables equal to 1 maximize the expression a_1a_2. Figure 4.1 illustrates the two bases which maximize the denominator expression a_1a_2 found in eq. 4.1. The other optimal solution to the maximization objective is $\{x_{12}, x_{13}, x_{21}\}$. The minimization of the original objective function will be the intersection between the solution space for the numerator and denominator objectives. The information about the maximization function will now constrain the minimization function to just these choices of decision variables. We can now write additional constraints which will cause one and only one of these sets of variables to be basic in the minimization objective. The numerator minimization objective, properly constrained, will become the minimum for the original objective.

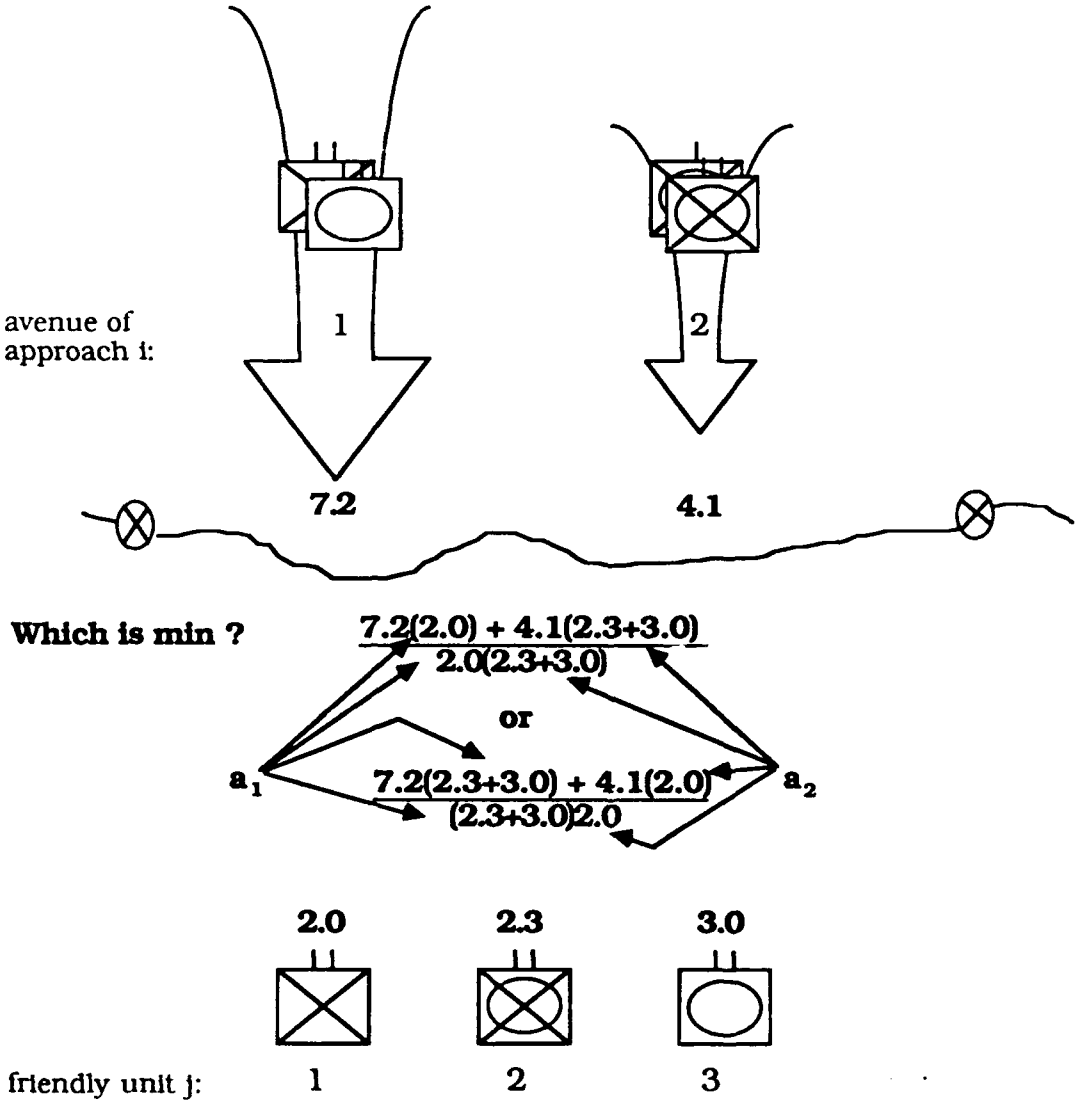


Figure 4.1: Example 1 Degeneracy

STEP 10: Three ($n!$ + 1) new constraints would restrict the minimization solution space to one of the maximization solutions. These constraints are:

$$x_{11} + x_{22} + x_{23} - 3y_1 = 0$$
$$x_{12} + x_{13} + x_{21} - 3y_2 = 0$$
$$y_1 + y_2 = 1$$

The coefficient, 3, on y_1 and y_2 in the first two constraints causes y_1 and y_2 to equal either 0 or 1 depending on which set of x_{ij}s are basic. Earlier it was stated either x_{11}, x_{22}, and x_{23} all equal 1 or x_{12}, x_{13}, and x_{21} all equal 1. Both sets of x_{ij}s cannot be chosen because the third constraint causes either y_1 or y_2 to equal 1. Thus the y_i acts as a "switch" by activating the decision variables contained in one the first two constraints to equal 1 at feasibility.

STEP 11: The redefined minimization problem becomes:

Min $21.6x_{21} + 16.56x_{22} + 14.4x_{23} + 12.3x_{11} + 9.43x_{12} + 8.2x_{13}$

Subject to:

$$x_{11} + x_{12} + x_{13} >= 1$$
$$x_{21} + x_{22} + x_{23} >= 1$$
$$x_{11} + x_{21} = 1$$
$$x_{12} + x_{22} = 1$$
$$x_{13} + x_{23} = 1$$

Additionally, from the maximization of the denominator problem:

$$
\begin{aligned}
x_{11} + x_{22} + x_{23} - 3y_1 &= 0 \\
x_{12} + x_{13} + x_{21} - 3y_2 &= 0 \\
y_1 + y_2 &= 1
\end{aligned}
$$

Now using a branch and bound minimization method, the above 0-1 formulation yields the following solution: The optimal basis is $\{x_{12}, x_{13}, x_{21}\}$. These variables yield a minimum objective value of **3.04** to the original objective function:

$$\frac{7.2}{a_1} + \frac{4.1}{a_2}$$

4.1.4 Conclusions

Even though this problem is small, the procedure is lengthy. In fact there are only six feasible solutions to this problem. To illustrate the growing complexity of this formulation, examine a slightly larger problem. Consider assigning five units opposite just three avenues of approach. Table 4.1 lists the types and numbers of constraints and variables generated in this kind of formulation.

Table 4.1: Types and Numbers of Constraints and Variables

Form	Type	#
$m+n$	assignment constraints	8
n	a_i equalities	3
m^2	double products with each double product having 3 constraints	75
m^3	triple products with each triple product having 4 constraints	600
	Total Number of Constraints:	**686**
mn	assignment variables	15
m^2	linear variable substitutions for double products	25
m^3	linear variable substitutions for triple products	125
$n!$	linear switch variables	6
	Total Number of Variables:	**171**

This fractional integer programming formulation would have 171 variables and 686 constraints and is still not the size that military staffs would typically need to analyze. A typical military problem found at division and corps level might contain 10 or more avenues of approach and 20 or more units to assign. The problem with 10 avenues would generate an overwhelming 10! linear switch variables. This problem, then, would contain over three and a half million linear switch variables alone. Clearly, the fractional integer program method may tax the limits of memory found in current day hardware and may, indeed, not yield a solution in the time allocated for analysis. A better, possibly more efficient, method is needed for larger problems. The Near-Minimal Algorithm demonstrated in Example 2 is such a method.

4.2 Example 2 (Use of the Near-Minimal Algorithm)

4.2.1 Problem Statement

Align, near optimally, friendly forces with combat power indices of 3.0, 2.1, 1.7, 3.5, and 2.3 against three known enemy avenues of approach with opposing combat power indices of 14.1, 8.6, and 10.7. Figure 4.2 graphically illustrates this example.

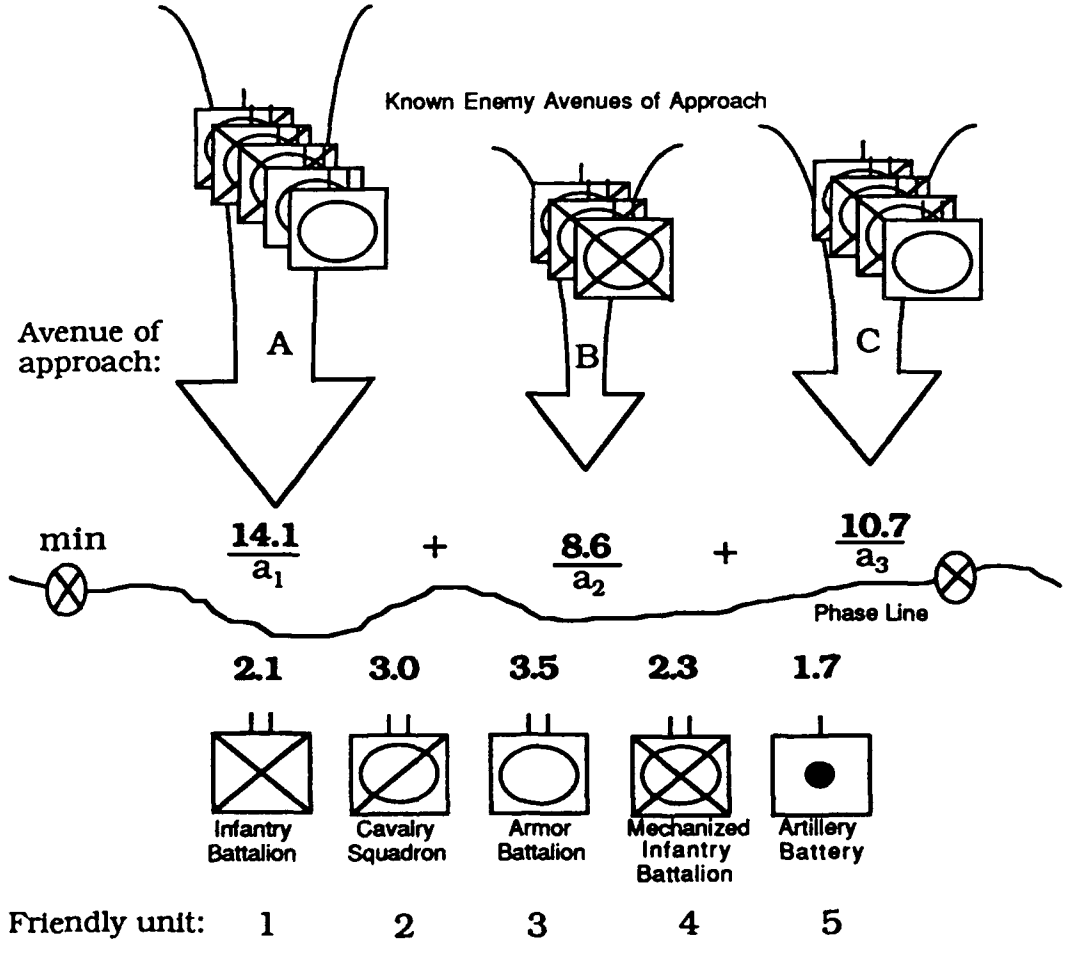


Figure 4.2: Example 2 Situation

4.2.2 Problem Discussion

We will choose the units for each avenue of approach such that we minimize the sum of ratios of the enemy's combat power indices to the units chosen to oppose it there. We will use the procedure in Chapter 3. Using this procedure, we can find a feasible, near-minimal solution in just five iterations of the algorithm. Each of the five iterations of the algorithm assigns a friendly unit and corresponds to a different row of Table 1. Table 2 records the assignments and aids in calculating the resulting force ratios.

For this example the optimal sum of the force ratios is 7.89.

4.2.3 Application of the Near-Minimal Algorithm

Worksheet Preparation
(refer to figure 4.3)

STEP 1: The indices for each avenue of approach were given in the problem statement. They are: 14.1, 8.6, and 10.7. These are recorded in row "red." The final two columns D and E are crossed out. They are not needed for this problem. There is a total of three avenues of approach in this problem. Record 3 in the "total entries" box.

STEP 2: Again, from the problem statement, we note the indices for the friendly units, in non-increasing order as: 3.5, 3.0, 2.3, 2.1, and 1.7. These are recorded in column "blue." This is a total of five friendly units. Record 5 in the "total entries" box.

STEP 3: Subtracting the two "total entries" establishes "m" as 2. Record 2 in box m. This is the maximum number of units that can be co-assigned to all avenues.

STEP 4: Let "u" equal 0 and "k" equal 1.

A Worksheet for Near Optimization of Force Ratios

(in the defense)

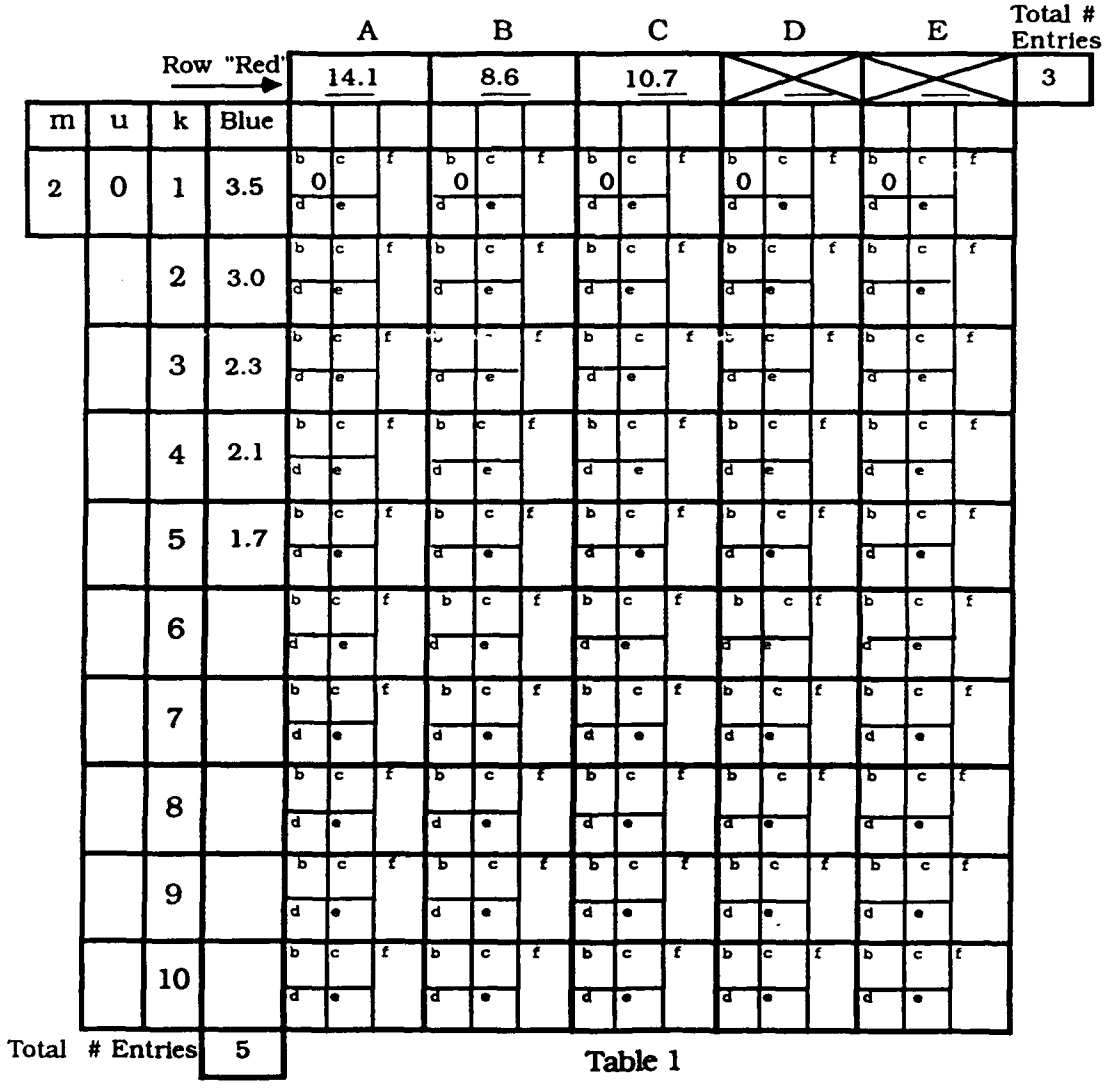


Figure 4.3: Algorithm Worksheet Preparation

Iteration 1
Worksheet Row 1
(refer to figure 4.4)

STEP 5: Add the entry in column "Blue" (3.5) to the each column's entry in box b and record in box c for each column.

STEP 6: Applying Rule 2, divide the entry in box c into the entry in row "red." Record in box e. All entries in box b are 0. Therefore, subtract the entry in box e from the entry in row "red" for each column. Record in box f.

STEP 7: Inspection of each entry in box f finds 10.08 as the greatest. Circle 10.08. The unit with potential 3.5 has its ***greatest marginal contribution*** toward minimizing the objective if it were assigned to avenue C. For column A only, add 3.5 (entry in column "blue" for row 1) to 0 and record in box b of row 2. For columns B and C record 0 (entry in box b of row 1) in box b of row 2.

STEP 8: We now record 3.5 (entry in column "blue" for row 1) in any box of column A (the column with 10.08 circled) in Table 2.

STEP 9: Since only one entry is listed in column A of Table 2 the counter u is unchanged. Record 0 in column u of row 2.

STEP 10: The value of u is still less than m. So we continue without violating feasibility.

STEP 11: We still have rows with entries in column "blue" that remain unfilled. So we continue.

STEP 12: Increment the row number by 1. We now work on row 2.

A Worksheet for Near Optimization of Force Ratios

(in the defense)

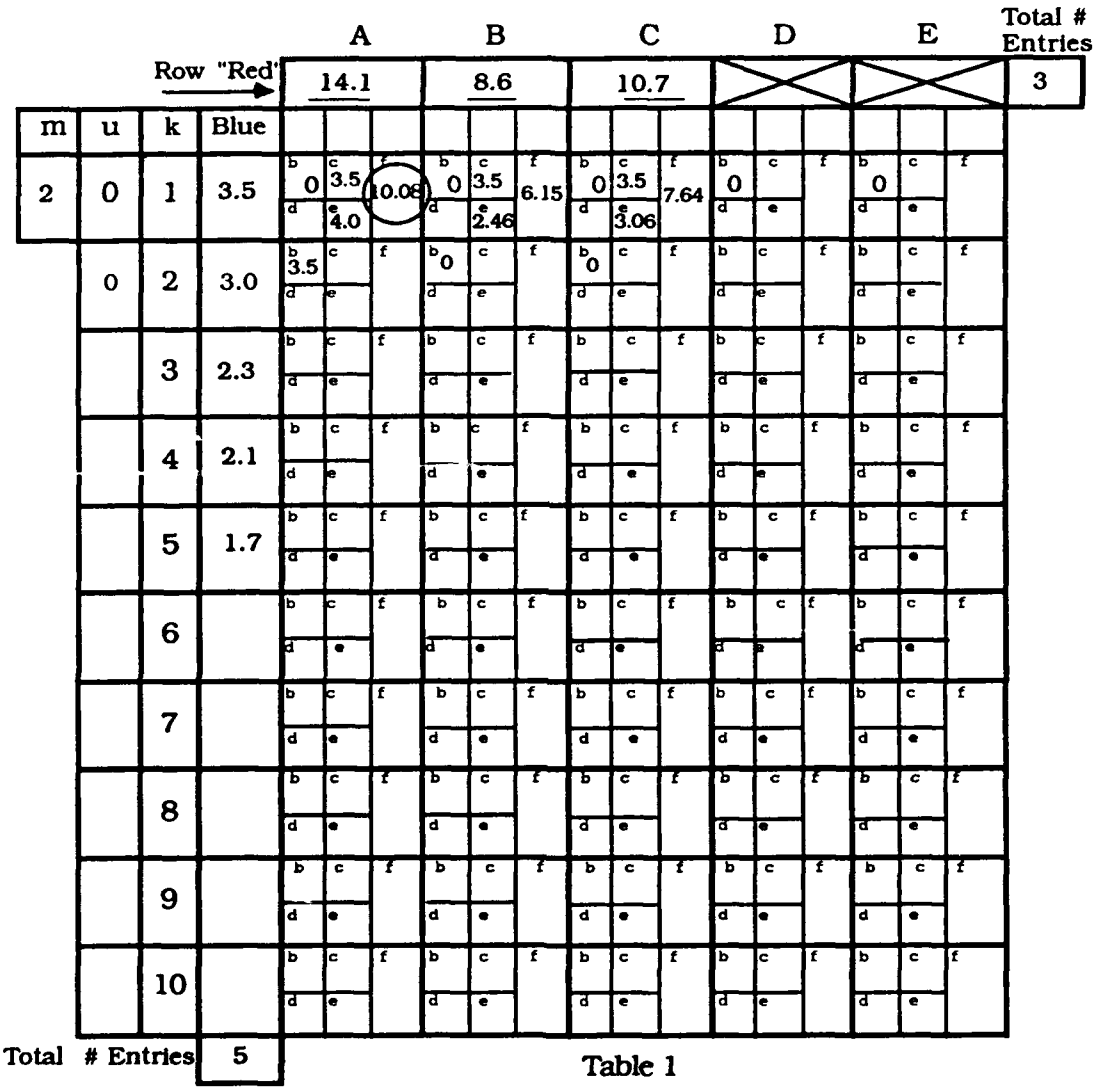


Figure 4.4: Algorithm Worksheet Iteration 1

Iteration 2
Worksheet Row 2
(refer to figure 4.5)

STEP 5: Add the entry in column "Blue" (3.0) to the each column's entry in box b and record in box c for each column.

STEP 6: Applying Rule 2, divide the entry in box c into the entry in row "red." Record in box e. All entries in box b are 0 except column A. Therefore, for columns B and C subtract the entry in box e from the entry in row "red" for each column. Record in box f. For column A divide the entry in box b into the entry in row "red" and record in box d. For column A only, subtract the entry in box e from the entry in box d. Record in box f.

STEP 7: Inspection of each entry in box f finds 7.14 as the greatest. Circle 7.14. The unit with potential 3.0 has its ***greatest marginal contribution*** toward minimizing the objective if it were assigned to avenue C. For column C only, add 3.0 (entry in column "blue" for row 2) to 0 and record in box b of row 3. For column A record 3.5 and for B record 0 (entry in box b of row 2) in box b of row 3.

STEP 8: We now record 3.0 (entry in column "blue" for row 2) in any box of column C (the column with 7.14 circled) in Table 2.

STEP 9: Since only one entry is listed in column C of Table 2 the counter u is unchanged. Record 0 in column u of row 3.

STEP 10: The value of u is still less than m. So we continue without violating feasibility.

STEP 11: We still have rows with entries in column "blue" that remain unfilled. So we continue.

STEP 12: Increment the row number by 1. We now work on row 3.

A Worksheet for Near Optimization of Force Ratios

(in the defense)

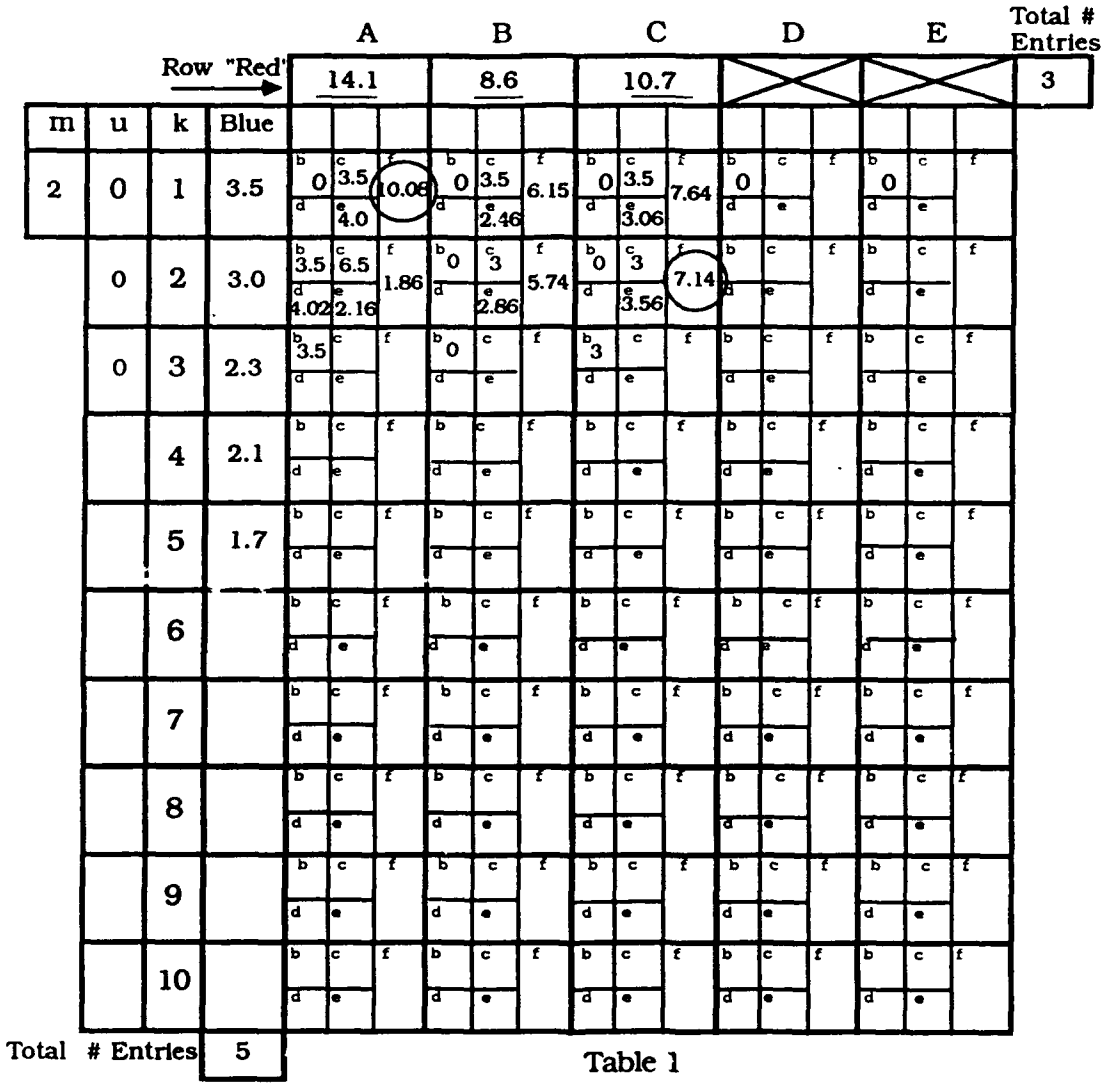


Figure 4.5: Algorithm Worksheet Iteration 2

Iteration 3
Worksheet Row 3
(refer to figure 4.6)

STEP 5: Add the entry in column "Blue" (2.3) to the each column's entry in box b and record in box c for each column.

STEP 6: Applying Rule 2, divide the entry in box c into the entry in row "red." Record in box e. Only column B's entry in box b is 0. Therefore, for column B subtract the entry in box e from the entry in row "red." Record in box f. For columns A and C divide the entry in box b into the entry in row "red" and record in box d. For columns A and C only, subtract the entry in box e from the entry in box d. Record in box f.

STEP 7: Inspection of each entry in box f finds 4.87 as the greatest. Circle 4.87. The unit with potential 2.3 has its ***greatest marginal contribution*** toward minimizing the objective if it were assigned to avenue B. For column B only, add 2.3 (entry in column "blue" for row 3) to 0 and record in box b of row 4. For column B record 2.3 and for A and C copy 3.5 and 3 (entries in box b of row 2) in box b of row 4 respectively.

STEP 8: We now record 2.3 (entry in column "blue" for row 3) in any box of column B (the column with 4.87 circled) in Table 2.

STEP 9: Since only one entry is listed in column B of Table 2 the counter u is unchanged. Record 0 in column u of row 4.

STEP 10: The value of u is still less than m. So we continue without violating feasibility.

STEP 11: We still have rows with entries in column "blue" that remain unfilled. So we continue.

STEP 12: Increment the row number by 1. We now work on row 4.

A Worksheet for Near Optimization of Force Ratios

(in the defense)

				A	B	C	D	E	Total # Entries
			Row "Red" →	14.1	8.6	10.7			3
m	u	k	Blue						
2	0	1	3.5	b 0 c 3.5 f 10.08 d e 4.0	b 0 c 3.5 f 6.15 d e 2.46	b 0 c 3.5 f 7.64 d e 3.06	b 0 c f d e	b 0 c f d e	
	0	2	3.0	b 3.5 c 6.5 f 1.86 d 4.02 e 2.16	b 0 c 3 f 5.74 d e 2.86	b 0 c 3 f 7.14 d e 3.56	b c f d e	b c f d e	
	0	3	2.3	b 3.5 c 5.8 f 1.59 d 4.02 e 2.43	b 0 c 2.3 f 4.87 d e 3.73	b 3 c 5.3 f 1.57 d 3.56 e 2.01	b c f d e	b c f d e	
	0	4	2.1	b 3.5 c f d e	b 2.3 c f d e	b 3 c f d e	b c f d e	b c f d e	
		5	1.7	b c f d e	b c f d e	b c f d e	b c f d e	b c f d e	
		6		b c f d e	b c f d e	b c f d e	b c f d e	b c f d e	
		7		b c f d e	b c f d e	b c f d e	b c f d e	b c f d e	
		8		b c f d e	b c f d e	b c f d e	b c f d e	b c f d e	
		9		b c f d e	b c f d e	b c f d e	b c f d e	b c f d e	
		10		b c f d e	b c f d e	b c f d e	b c f d e	b c f d e	
Total # Entries			5						

Table 1

	A	B	C	D	E	
	3.5	2.3	3.0			
Total Row Ratio						

Table 2 Total

Figure 4.6: Algorithm Worksheet Iteration 3

Iteration 4
Worksheet Row 4
(refer to figure 4.7)

STEP 5: Add the entry in column "Blue" (2.1) to the each column's entry in box b and record in box c for each column.

STEP 6: Applying Rule 2, divide the entry in box c into the entry in row "red." Record in box e. All columns have non-zero entries in box b. For all columns divide the entry in box b into the entry in row "red" and record in box d. For all columns, subtract the entry in box e from the entry in box d. Record in box f.

STEP 7: Inspection of each entry in box f finds 1.79 as the greatest. Circle 1.79. The unit with potential 2.1 has its ***greatest marginal contribution*** toward minimizing the objective if it were assigned to avenue B. For column B only, add 2.1 (entry in column "blue" for row 4) to 2.3 (entry in box b for column B) and record in box b of row 5. For column B record 4.4 and for A and C copy 3.5 and 3 (entries in box b of row 4) in box b of row 5 respectively.

STEP 8: We now record 2.1 (entry in column "blue" for row 4) in any remaining box of column B (the column with 1.79 circled) in Table 2.

STEP 9: Since more than one entry is listed in column B of Table 2 the counter u is incremented by 1. Record 1 in column u of row 5.

STEP 10: The value of u is still less than m. So we continue without violating feasibility.

STEP 11: We still have rows with entries in column "blue" that remain unfilled. So we continue.

STEP 12: Increment the row number by 1. We now work on row 5.

A Worksheet for Near Optimization of Force Ratios

(in the defense)

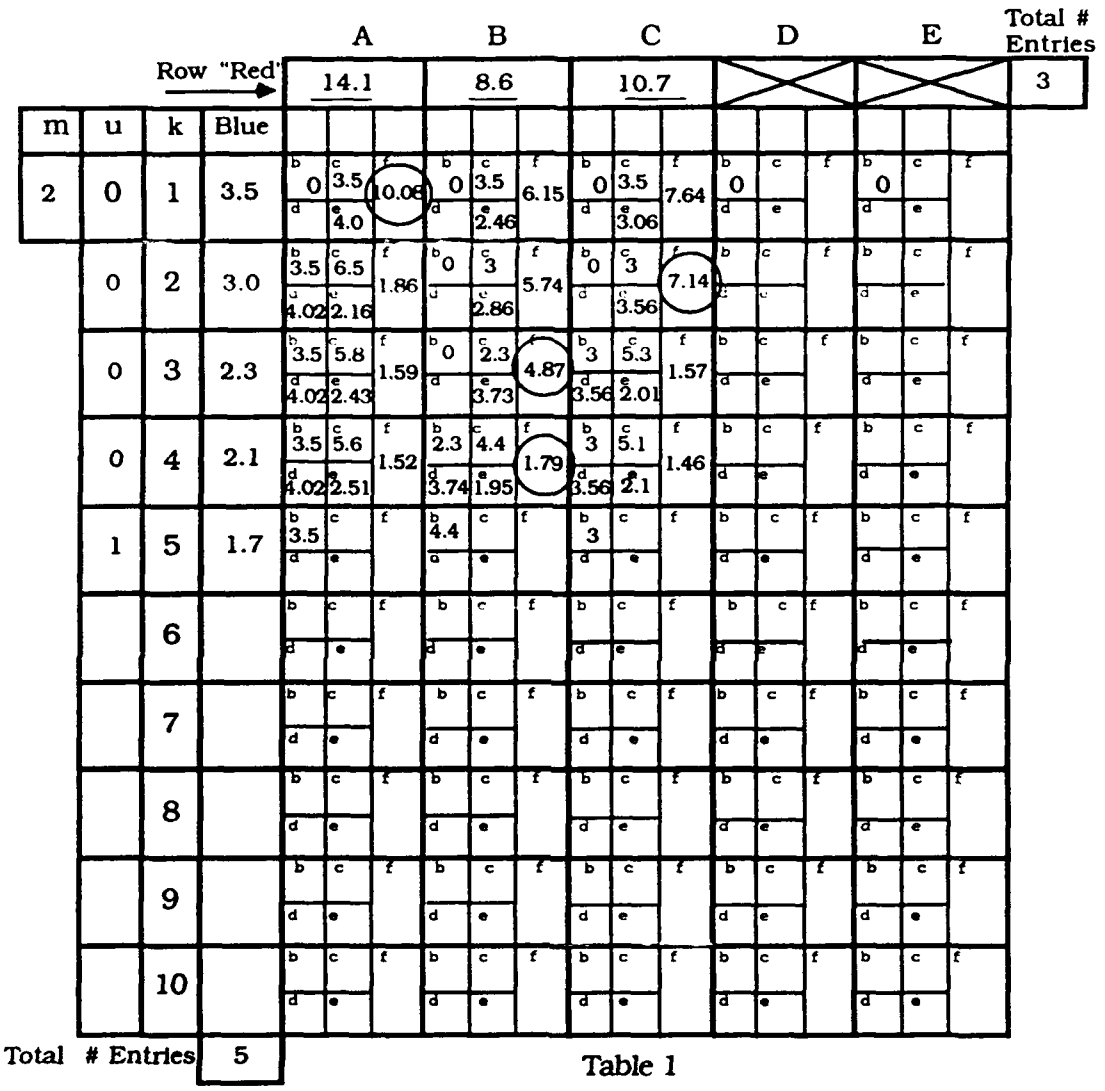


Figure 4.7: Algorithm Worksheet Iteration 4

Iteration 5
Worksheet Row 5 and Calculation of Force Ratios
(refer to figures 4.8 and 4.9)

STEP 5: Add the entry in column "Blue" (1.7) to the each column's entry in box b and record in box c for each column.

STEP 6: Applying Rule 2, divide the entry in box c into the entry in row "red." Record in box e. All columns have non-zero entries in box b. For all columns divide the entry in box b into the entry in row "red" and record in box d. For all columns, subtract the entry in box e from the entry in box d. Record in box f.

STEP 7: Inspection of each entry in box f finds 1.31 as the greatest. Circle 1.31. The unit with potential 1.7 has its ***greatest marginal contribution*** toward minimizing the objective if it were assigned to avenue A. For column A only, add 1.7 (entry in column "blue" for row 5) to 3.5 (entry in box b for column B) and record in box b of row 6. For column A record 5.2 and for B and C copy 4.4 and 3 (entries in box b of row 5) in box b of row 6 respectively.

STEP 8: We now record 1.7 (entry in column "blue" for row 5) in any remaining box of column A (the column with 1.31 circled) in Table 2.

STEP 9: Since more than one entry is listed in column A of Table 2 the counter u is incremented by 1. Record 2 in column u of row 6.

STEP 10: The value of u is now equal to m. No further calculations are necessary for columns A, B, and C (columns with circled entries in box f). Cross out the remaining portions of columns A, B, and C.

STEP 11: We have no rows remaining with entries in column "blue" that remain unfilled. Skip Step 12 and go directly to Step 13.

STEP 13: We now shift to Table 2 for calculation of force ratios. Sum the entries

for each column in Table 2 and record in row "total." Divide each column's total into its entry in row "red" of Table 1. Record the results, 2.71, 1.95, and 3.56 in row "ratio." These are the near-minimal force ratio's for each avenue of approach.

STEP 14: Sum the entries in row "ratio." Record the result, 8.22, in box "total." This sum is the near-minimal sum to the fractional integer programming problem. Assign units with combat power indices of 3.5 and 1.7 to avenue of approach A. Assign units with combat power indices 2.3 and 2.1 to avenue of approach B. Assign the unit with combat power index 3.0 to avenue of approach C. Figure 4.9 illustrates the near-optimal assignment of units. This ends the procedure.

4.2.4 Conclusions

Although the Near-Minimal Algorithm is simple, it is computation intense. The worksheet can handle a problem with as many as five avenues of approach and 10 units to assign. Example 3 demonstrates use of the worksheet for a problem with five avenues and seven units.

A Worksheet for Near Optimization of Force Ratios

(in the defense)

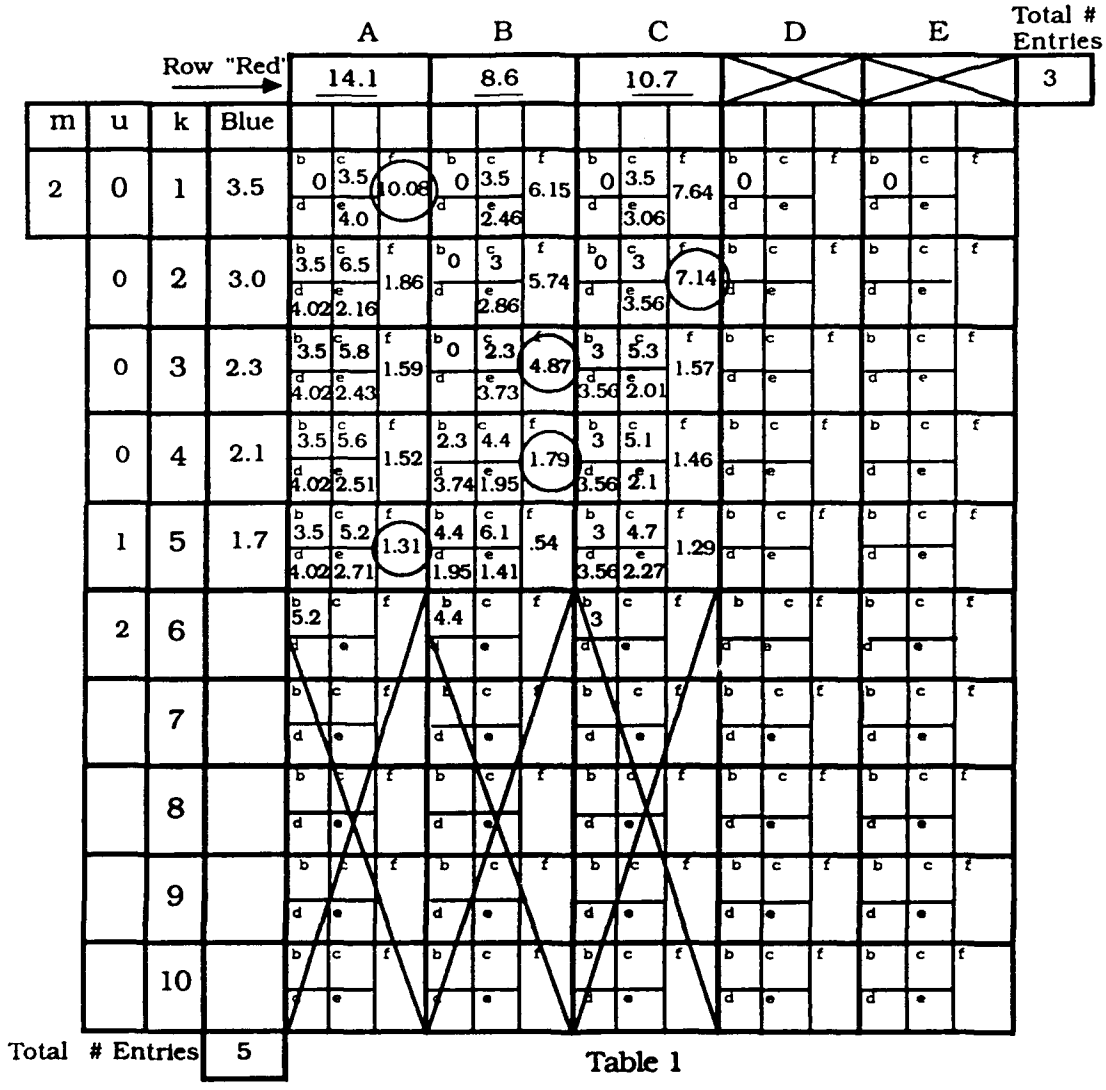


Table 1

	A	B	C	D	E	
	3.5	2.3	3.0			
	1.7	2.1				
Total	5.2	4.4	3.0			
Row Ratio	2.71	1.95	3.56			8.22

Table 2 Total

Figure 4.8: Algorithm Worksheet Iteration 5

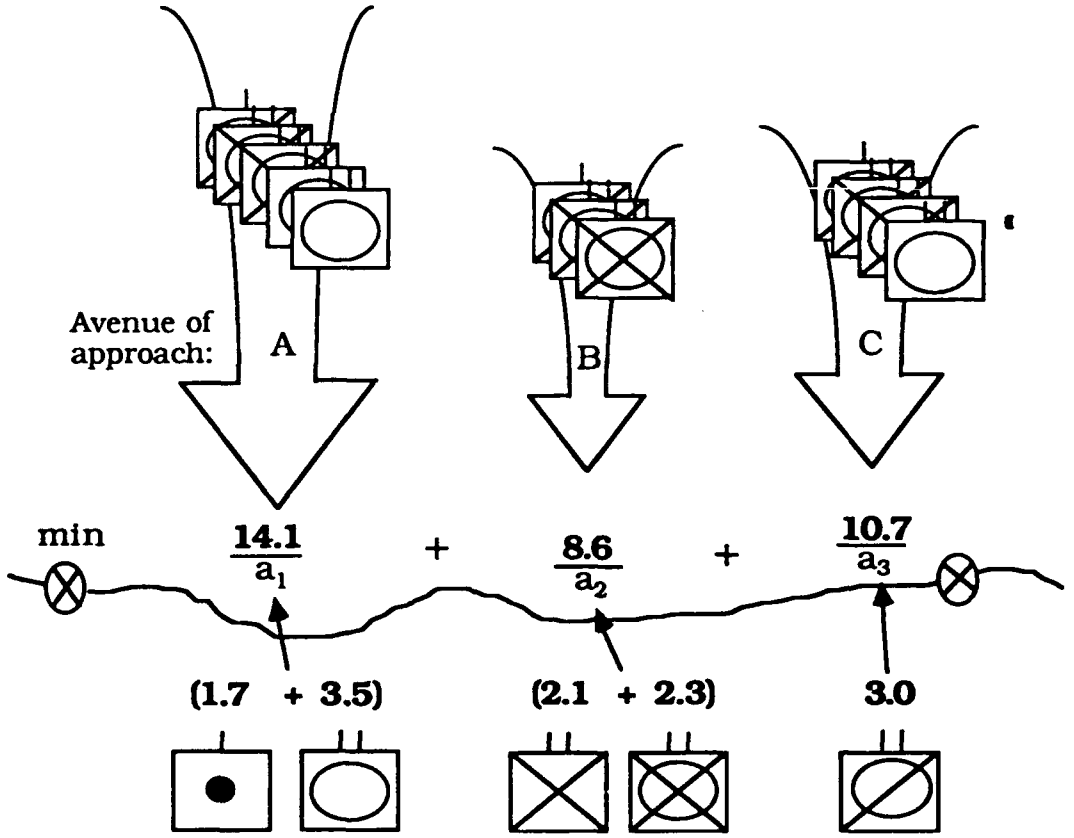


Figure 4.9: Example 2 Final Assignments

4.3 Example 3 (Summarized Use of the Near-Minimal Algorithm)

4.3.1 Problem Statement

Align, near optimally, friendly forces with combat power indices of 3.0, 2.0, 2.0, 3.0, 2.0, 2.0 and 2.0 against five known enemy avenues of approach with opposing combat power indices of 4.9, 8.0, 3.5, 12.9 and 3.5. The graphic illustration for this example is shown is figure 4.10.

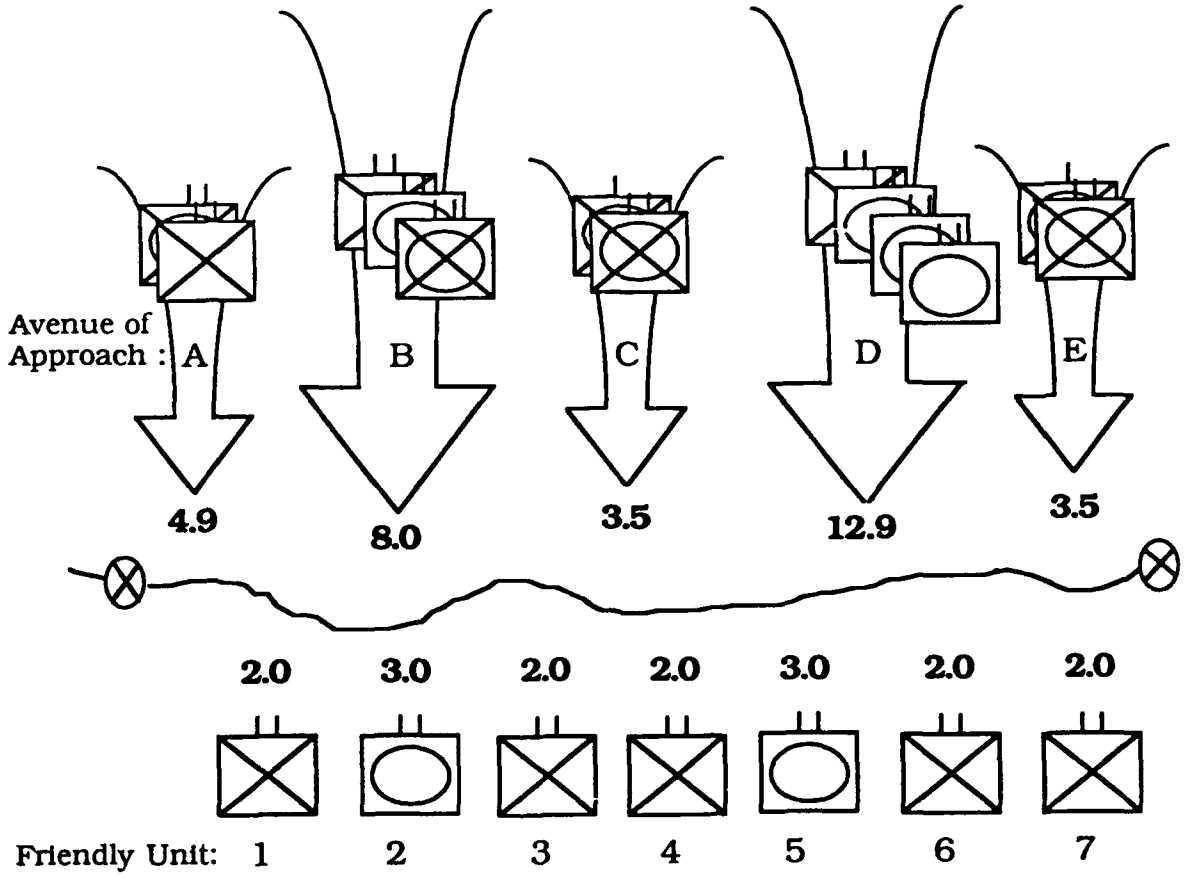


Figure 4.10: Examples 3 and 4 Situation

4.3.2 Problem Discussion

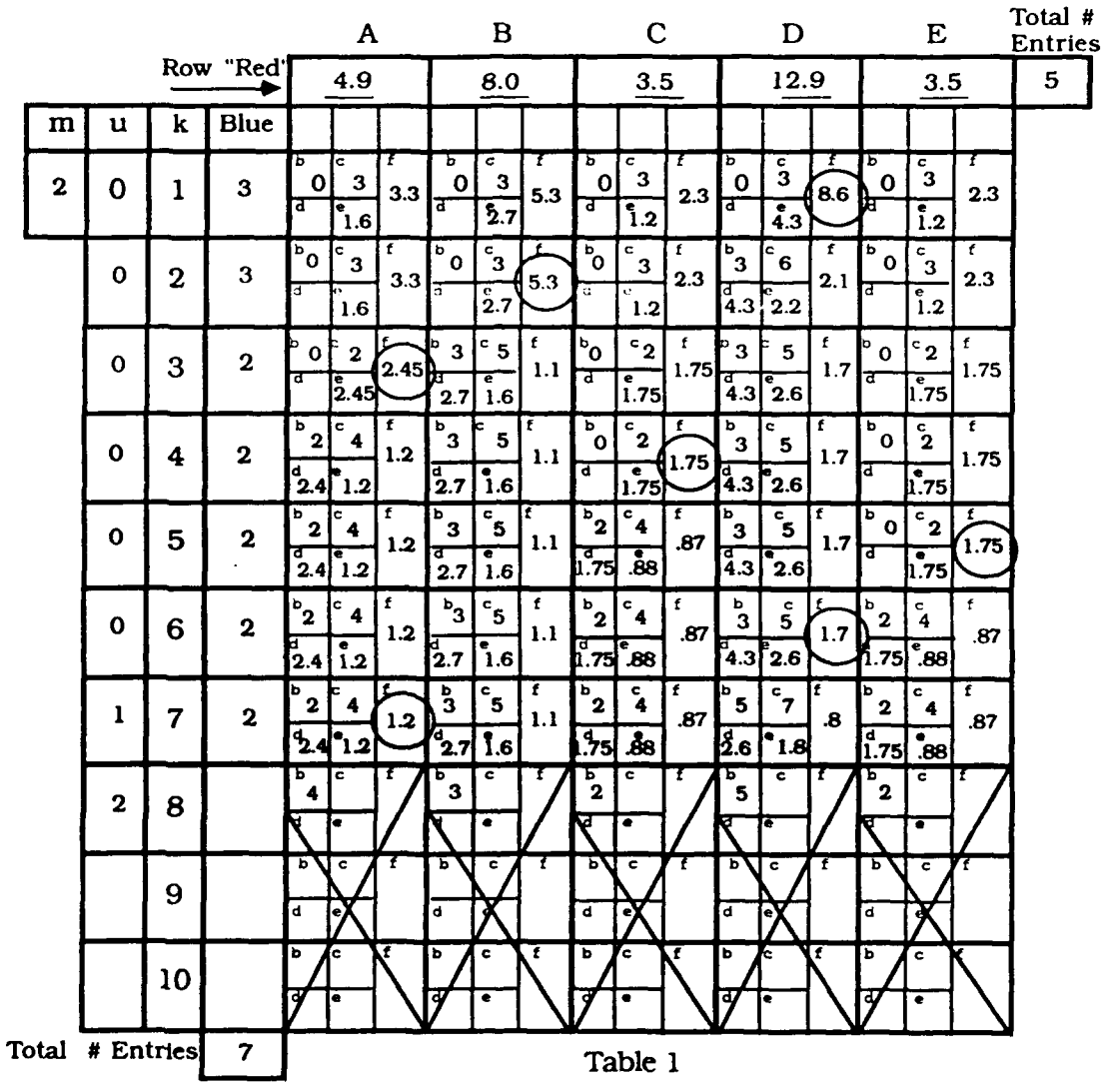
Again, we will choose the units for each avenue of approach such that we minimize the sum of ratios of the enemy's combat power indices to the units chosen to oppose it there. We will use the procedure and worksheet in Chapter 3. Using this procedure, we can find a feasible, near-minimal solution in just seven iterations of the algorithm. The completed worksheet for this problem is shown in figure 4.11.

4.3.3 Conclusions

The computations for these larger problems are repetitive, time consuming, and tiresome. In fact, for problems larger than five avenues and 10 units another worksheet would be necessary to record the necessary computations. Aside from this shortcoming, an analyst could easily make an error during any one of the many calculations. Worse yet, an incorrect decision may result concerning unit assignments. The computer program demonstrated in the next example requires minimal input to define the problem and completes the worksheet automatically and very quickly.

A Worksheet for Near Optimization of Force Ratios

(in the defense)



m	u	k	Blue	A	B	C	D	E	Total # Entries
			Row "Red" →	4.9	8.0	3.5	12.9	3.5	5
2	0	1	3	b 0 c 3 d e 1.6 f 3.3	b 0 c 3 d e 2.7 f 5.3	b 0 c 3 d e 1.2 f 2.3	b 0 c 3 d e 4.3 f 8.6	b 0 c 3 d e 1.2 f 2.3	
	0	2	3	b 0 c 3 d e 1.6 f 3.3	b 0 c 3 d e 2.7 f 5.3	b 0 c 3 d e 1.2 f 2.3	b 3 c 6 d 4.3 e 2.2 f 2.1	b 0 c 3 d e 1.2 f 2.3	
	0	3	2	b 0 c 2 d e 2.45 f 2.45	b 3 c 5 d 2.7 e 1.6 f 1.1	b 0 c 2 d e 1.75 f 1.75	b 3 c 5 d 4.3 e 2.6 f 1.7	b 0 c 2 d e 1.75 f 1.75	
	0	4	2	b 2 c 4 d 2.4 e 1.2 f 1.2	b 3 c 5 d 2.7 e 1.6 f 1.1	b 0 c 2 d e 1.75 f 1.75	b 3 c 5 d 4.3 e 2.6 f 1.7	b 0 c 2 d e 1.75 f 1.75	
	0	5	2	b 2 c 4 d 2.4 e 1.2 f 1.2	b 3 c 5 d 2.7 e 1.6 f 1.1	b 2 c 4 d 1.75 e .88 f .87	b 3 c 5 d 4.3 e 2.6 f 1.7	b 0 c 2 d e 1.75 f 1.75	
	0	6	2	b 2 c 4 d 2.4 e 1.2 f 1.2	b 3 c 5 d 2.7 e 1.6 f 1.1	b 2 c 4 d 1.75 e .88 f .87	b 3 c 5 d 4.3 e 2.6 f 1.7	b 2 c 4 d 1.75 e .88 f .87	
	1	7	2	b 2 c 4 d 2.4 e 1.2 f 1.2	b 3 c 5 d 2.7 e 1.6 f 1.1	b 2 c 4 d 1.75 e .88 f .87	b 5 c 7 d 2.6 e 1.8 f .8	b 2 c 4 d 1.75 e .88 f .87	
	2	8		b 4 c d e f	b 3 c d e f	b 2 c d e f	b 5 c d e f	b 2 c d e f	
		9		b c d e f	b c d e f	b c d e f	b c d e f	b c d e f	
		10		b c d e f	b c d e f	b c d e f	b c d e f	b c d e f	
Total # Entries			7						

Table 1

	A	B	C	D	E	
	2	3	2	3	2	
	2			2		
Total	4	3	2	5	2	
Row Ratio	1.75	2.67	1.23	2.58	1.75	9.97

Table 2 — Total

Figure 4.11: Example 3 Completed Worksheet

4.4 Example 4 (Using the Analogous Computer Program)

4.4.1 Problem Statement

Align, near optimally, friendly forces with combat power indices of 3.0, 2.0, 2.0, 3.0, 2.0, 2.0 and 2.0 against five known enemy avenues of approach with opposing combat power indices of 4.9, 8.0, 3.5, 12.9 and 3.5.

4.4.2 Problem Discussion

This situation is identical to example 3 and is graphically illustrated in figure 4.10. However, in this example we will execute the algorithm with the aid of a computer. The software will assign units for each avenue of approach such that it nearly minimizes the sum of the ratios of the enemy combat power indices over the sum of the friendly unit indices chosen to oppose it. The C code for the computer program we will use is at Appendix E. This program executes the exact procedure illustrated manually in examples 2 and 3. Any IBM compatible computer that operates on MS-DOS can execute the compiled version of the program. With the aid of this program, we can find a feasible, near-minimal solution in less than half a second. The user and software interfacing is shown in figure 4.12.

User-Software Interface

```
**********************************************************
        This software aligns tactical forces
                in the defense.

        It can assign up to 20 friendly units
         across 20 enemy avenues of approach.
**********************************************************

How many enemy avenues of approach are known?  (Please enter
 an integer between 1 and 20) 5

Enter 5 enemy combat power indices. Separate each index
 with a space or a <return>. The first one should correspond
 with avenue of approach A; the second with B; etc.
4.9
8
3.5
12.9
3.5
The enemy combat power indices you read in are as follows:
4.90, 8.00, 3.50, 12.90, 3.50.

How many friendly units are available for deployment? 7

Enter 7 friendly combat power indices. Separate each
 with a space or a <return>. Each one should correspond
 to a different friendly unit.
2  3  2  2  2  3  2
The friendly combat power indices you read in are as follows:
2.00, 3.00, 2.00, 2.00, 2.00, 3.00, 2.00.

 A near-optimal alignment of friendly units is as follows:

 Assign combat power index 3.00 to avenue of approach D.
 Assign combat power index 3.00 to avenue of approach B.
 Assign combat power index 2.00 to avenue of approach A.
 Assign combat power index 2.00 to avenue of approach E.
 Assign combat power index 2.00 to avenue of approach C.
 Assign combat power index 2.00 to avenue of approach D.
 Assign combat power index 2.00 to avenue of approach A.

 The Force Ratios are:

 A:  1.23;    B:  2.67;    C:  1.75;    D:  2.58;    E:  1.75;

-------------------------------------------
*    The near-optimal solution is 9.97.   *
-------------------------------------------
```

Figure 4.12: Computer Program Example

Chapter 5

CONCLUSION AND SUGGESTIONS FOR FURTHER RESEARCH

> In order to fight the battle successfully the commander has to find out what is going on, *decide* what to do about it, tell somebody what to do, then keep track of how the battle is going...
>
> General Starry 1981

5.1 Conclusion

It is in the decision process that commanders are most often called upon to exercise their awesome responsibility. It is the commander who is given all available and relevant information to the situation. It is the commander who most often *decides* what to do. However, should time permit, his or her decision is often based on the analysis of raw facts at some lower level. It is the commander's staff which gleefully dissects the situation and presents the timely, pertinent facts necessary for the commander's decision. Tactically, the decision of how to defend terrain is one of the most critical a commander can make. A ground force must first establish an adequate defense before launching any other type of operation. A defense is necessary to exist on the ground. In peacetime and in wartime ground forces must defend themselves first. To fail in the defense could cause disastrous consequences for a tactical force and possibly our nation.

This thesis is aimed at strengthening the staff's analysis of the defense. The current method used in the Army is deficient in that it does not incorporate in its formulation a critical evaluation criterion. Force ratios are important in the analysis of defense but the defense is not formulated with a detailed inquiry as to what ratio of force

is the best for the tactical situation. This thesis presents a method in which a detailed assessment of force ratios can be affected very quickly.

It should be noted that a very good solution is obtained in less than half a second of run time on an IBM 286 compatible running at 12MHz. This is especially significant when compared to present integer methods which, after correct formulation, could run for several hours or more. The algorithm presented in this thesis produces a substantial time savings. It has also been shown in Appendix D that the time savings is at the expense of very little mathematically. The solution is very near the minimal solution to the fractional integer program.

After applying the algorithm, the resulting assignment of tactical forces could function as a standard in which the staff could guide their own estimate of the situation. As a result the scope of analysis is reduced. Coupled, too, with the substantial time savings of the Algorithm, staff officers previously detailed to analyze force ratios are freed to conduct other, possibly indispensable, planning functions. The critical nature of the ensuing decision requires the staff to analyze the situation as best they possibly can. This thesis tremendously enhances the analysis of a defensive situation for both the staff and the commander.

5.2 Suggestions for Further Research

Although the method presented in this thesis is quick and dirty, some improvements may be desirable. Specifically, the algorithm can be improved mathematically and the packaging of the software is not as friendly as it needs to be for general, unlimited use by occasional computer users.

At first, the reader may hypothesize that some obvious generalizations and

shortcuts may be in order to shorten the rather repetitive manual method presented in Chapters 3 and 4 of this thesis. The author cautions the reader against hasty inferences. The reader should note that the examples in Chapter 4 are elementary sample problems chosen for their simplicity and ease of illustration. Not all situations bear the seemingly apparent generalizations that these examples tend to radiate. For example, it is not always the case that the first n units are each assigned to a different avenue of approach. If this were universally the case, then this procedure could be simplified. Further research may be in order to appropriately simplify the manual procedure without damaging the impact of the logic contained within.

This author conjectured a result in Chapter 3 during formulation of the fractional integer problem solving procedure which needs to be proved. In this procedure it was believed that if the denominator of an objective fraction were fixed at a feasible maximum, then the numerator searched for a feasible minimum, the resulting fraction is the minimum for the fractional objective. This author worked numerous examples applicable to this thesis and maintains this is a valid claim. However, the author presents no formal mathematical proof of this result. The fractional integer method contained in this thesis certainly should contain a proof of this result to legitimately claim optimality from the method. This is a topic for further research.

Mathematically, there is a defect in the near minimal algorithm. Fortunately, this defect will seldom, if at all, occur in practice because of the way the military problem is defined. Under certain conditions the algorithm proceeds and reaches a solution which is feasible but significantly removed from optimality. This situation can occur whenever friendly units are assigned a combat power index of 1. Here's why: The essence of the algorithm is the maximization of a differential between an original number (an enemy

combat power index) and a new number derived from dividing a friendly combat power index into this original number. When the number 1 is divided into any number the result is the number itself. When this situation occurs, the difference between the original number and the new number is 0. The assignment decision of this index to a particular avenue of approach is built on choosing a maximum difference. Since it is this difference which is being maximized, any other positive difference will always be greater. As a result, we will not see an assignment of a unit with index 1 until feasibility is critical. Under certain circumstances it is undesirable to assign a unit with index 1 so late in the algorithm. The resulting sum to the fractional integer objective may not be near-minimal. Fortunately and decidedly, the military problem is carefully developed by assigning the index 1 to an enemy unit as the relative standard. The author does not consider this potential deficiency a showstopper in any military application of this procedure. This is an area which should be further researched.

Another area of further research is in the packaging of the analogous computer program. The author taught himself the C language and admits to any deficiencies in the efficiency and friendliness of the code. Making the program more user friendly is certainly worthwhile. Also of interest could be the output itself. Currently, graphics is not used in presenting assignments of units or force ratios. Indeed, commanders and staffs would be less intimidated by graphical output. Ideally the graphics could employ current Army symbols and control measures and might even be a three-dimensional representation of the battlefield itself.

Combining force ratios with other relevant decision criteria is of interest, too. For example, integrating terrain limitations and lateral mobility potential for different units would extend military practicality. Terrain limitations would add a set of binding

constraints mathematically. Lateral mobility could be incorporated in a risk type analysis before final assignments of units. Both of these additional criteria are important but not addressed in this thesis. The decision to assign units to terrain is limited by geography. Certainly, there is a limit to the number of units that can be assigned to particular terrain (terrain limitations). Equally as important is how quickly we can displace to other terrain should the situation require it (lateral mobility). Coupling these two additional criteria with force ratios is an area for further research.

Finally, a weighting scheme that prioritizes the potential threat or its likelihood could be useful. If this were performed, the resulting assignment of units may be more reflective of the actual decision criteria. Rarely would an analyst consider the threat equally likely across all avenues of approach. Interestingly, too, the current Army force ratio calculation worksheet does not include a weighting scheme. Although the method in this thesis emulates the current Army worksheet in this manner, further research could benefit both procedures.

SELECTED BIBLIOGRAPHY

Gottfried, Byron S. 1990. Programming with C., Schaum Outline Series. New York: McGraw-Hill Publishing Co.

Graduate School, Colorado School of Mines. 1987. "The Thesis Writer's Handbook." Golden, CO.

Hu, T. C. 1982. Combinatorial Algorithms. Reading, Massachusetts: Addison-Wesley.

Long, Clyde L. Synchronization of Combat Power at the Task Force Level: Defining a Planning Methodology. A Master's Thesis presented to the Faculty of the U.S. Command and General Staff College, Fort Leavenworth, KS, 2 June 1989.

Megiddo, Nimrod. 1979. Combinatorial Optimization with Rational Objective Functions. Mathematics of Operations Research. 4: 414-424.

Mott, Joe L., Kandal Abraham, and Baker, Theodore P. 1986. Discrete Mathematics for Computer Scientists & Mathematicians. 2d ed. Englewood Cliffs, NJ: Printice-Hall.

Pamperl, LTC, 1990. Private Communications.

Press, William H., Et Al. 1988. Numerical Recipes in C, The Art of Scientific Computing. Cambridge: Cambridge University Press.

Shirron, W. Edward. An Optimum Method of Wargaming a Tactical and Operational Course of Action as an Integral Part of the Corps Commander's and G3's Estimate of the Situation in a Time Compressed Environment. A Master's Thesis presented to the Faculty of the U.S. Command and General Staff College, Fort Leavenworth, KS, 1984.

Tucker, Alan. 1980. Applied Combinatorics. New York: John Wiley & Sons.

U.S. Army Command and General Staff College, Fort Leavenworth, KS. RB 100-9, A Guide to the Application of the Estimate of the Situation in Combat Operations. September, 1983.

_____. ST 100-9, The Command Estimate. July 1989.

_____. ST 100-3, Battle Book. 1 April 1989.

_____. "School Example Division Operation Estimate. Preparation for Combat Operations Exercise." Section II, Lesson 12: 95-140. July 1990.

U.S. Department of the Army, Washington, D.C. FM 100-2-1, The Soviet Army Operations and Tactics. 16 July 1984.

_____. FM 100-2-3, The Soviet Army Troops, Organization and Equipment. 16 July 1984.

Waite, Michael., Et Al. 1990. Microsoft Quick C Programming. 2d ed. Redmond, WA: Microsoft Press.

_____. 1990. C: Step-by-Step. 2d ed. Carmel, IN: Howard W. Sams & Co.

Woolsey, R. E. D. Colorado School of Mines, Golden, CO, "Class Notes in Integer Programming," January 1990.

_____. 1990. Private Communications.

Appendix A

SOLUTION SET SIZES

The table below lists the total number of solutions possible for different size problems. Combinatoric occupancy theory was used to formulate the possible solution set sizes.

Table A-1: Combinatoric Solution Set Sizes

Avenues of Approach	Number of Friendly Units	Formulation	Possible Solutions
2	3	$\binom{3}{2}\binom{1}{1}\binom{2}{1}$	6
2	4	$\binom{4}{3}\binom{1}{1}\binom{2}{1}+\binom{4}{2}\binom{2}{2}\binom{2}{2}$	14
3	4	$\binom{4}{2}\binom{2}{1}\binom{1}{1}\binom{3}{1}$	36
3	5	$\binom{5}{3}\binom{2}{1}\binom{3}{1}\binom{1}{1}+\binom{5}{2}\binom{3}{2}\binom{1}{1}\binom{3}{2}$	150
4	6	$\binom{6}{3}\binom{3}{1}\binom{2}{1}\binom{1}{1}\binom{4}{1}+\binom{6}{2}\binom{4}{2}\binom{2}{1}\binom{1}{1}\binom{4}{2}$	1560
4	7	$\binom{7}{4}\binom{3}{1}\binom{2}{1}\binom{1}{1}\binom{4}{1}+\binom{7}{2}\binom{5}{2}\binom{3}{2}\binom{1}{1}\binom{4}{3}+\binom{7}{3}\binom{4}{2}\binom{2}{1}\binom{1}{1}\binom{4}{2}$	5880
5	7	$\binom{7}{3}\binom{4}{1}\binom{3}{1}\binom{2}{1}\binom{1}{1}\binom{5}{1}+\binom{7}{2}\binom{5}{2}\binom{3}{1}\binom{2}{1}\binom{1}{1}\binom{5}{2}$	16800
5	8	$\binom{8}{4}\binom{4}{1}\binom{3}{1}\binom{2}{1}\binom{1}{1}\binom{5}{1}+\binom{8}{3}\binom{5}{2}\binom{3}{1}\binom{2}{1}\binom{1}{1}\binom{5}{2}+\binom{8}{2}\binom{6}{2}\binom{4}{2}\binom{2}{1}\binom{1}{1}\binom{5}{3}$	92400

Appendix B
BLANK FORMS

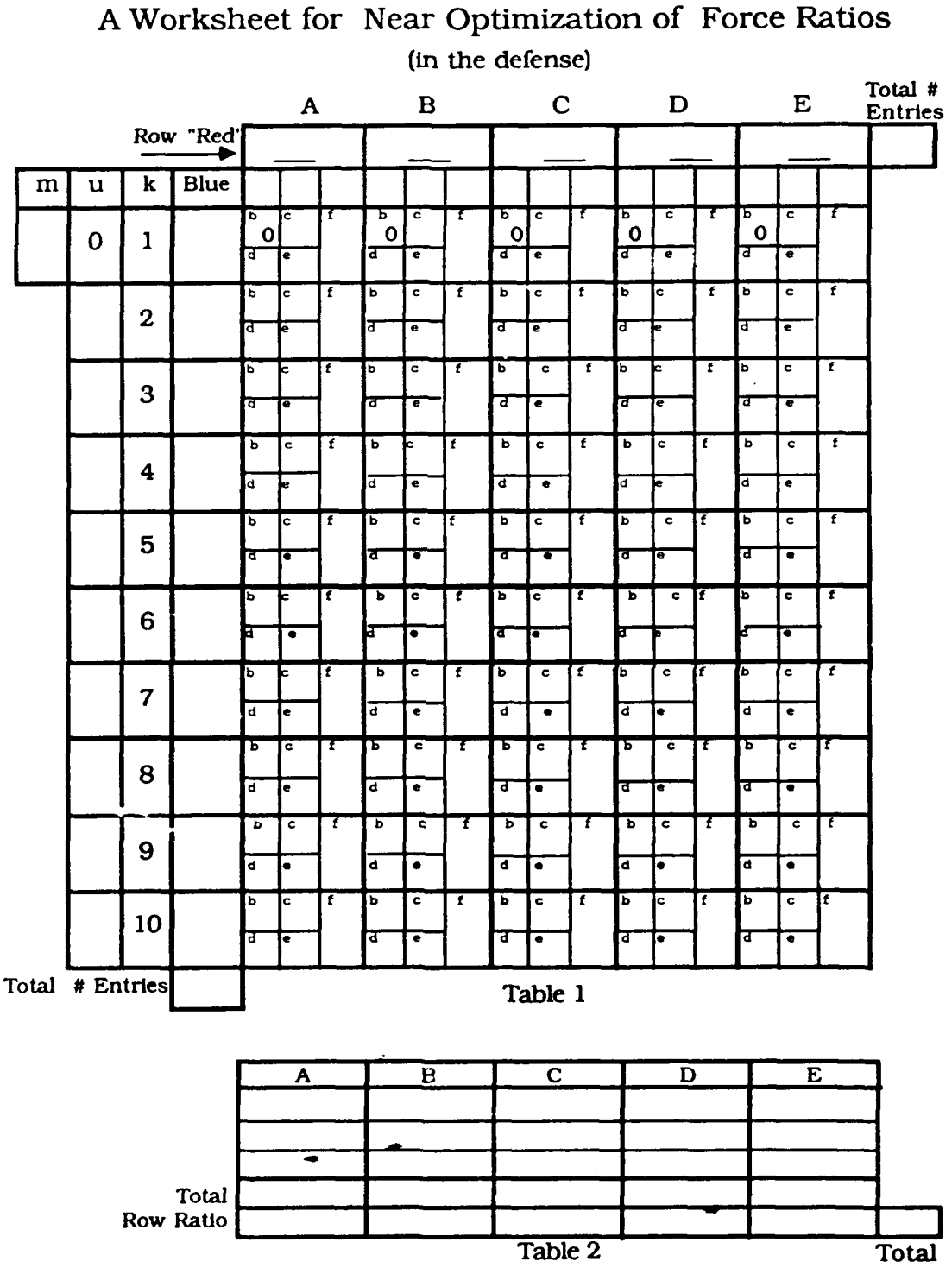


Figure B-1: Near-Minimal Algorithm Worksheet

ALLOCATION OF FORCES WORKSHEET

ALLOCATION STEPS: (All calculations rounded to nearest tenth)

A. Indicate avenue of approach letters.
B. Indicate threat echelons.
C. Display situation template.
D. Calculate enemy battalion equivalents.
E. Array and adjust friendly forces, allocating against enemy echelons.
F. Compute force ratios (FR).

	Brigade allocates against	Division allocates against	Corps allocates against
CFA 1:6/1:3	Divisions 1st Echelon (D1E)		
M B A 1:3	Regiments 1st Echelon (R1E)	D1E	Army's 1st Echelon (A1E)
	Regiments 1st & 2d Echelon (R12E)	Divisions 1st & 2d Echelon (D12E)	Army's 1st & 2d Echelon (A12E)
Res 1:3	D12E	A12E	Fronts 1st & 2d Echelon (F12E)

	(___) 2d Ech	
(___) 1st Ech	(___) 2d Ech	
	(___) 1st Ech	

Ave Letter	

CFA	1:6/1:3 vs D1E	
M B A	1:3 vs (___) 1E	
	1:3 (cum) vs (___) 1, 2, E	
RES	1:3 (cum) vs (___) 1, 2, E	
FR =	(___) 1E 1:	
	(___) 1E 1:	

86-(F626)-3352—15,000—10 Sep 86

Figure B-2: FORM 86-(F626)-3352

Appendix C

SAMPLE COMPUTER RUNS

```
C:\QC2>aofa4
   ************************************************************
           This software aligns tactical forces
                      in the defense.

           It can assign up to 20 friendly units
            across 20 enemy avenues of approach.
   ************************************************************

How many enemy avenues of approach are known?  (Please enter
 an integer between 1 and 20) 2

Enter 2 enemy combat power indices. Separate each index
 with a space or a <return>. The first one should correspond
 with avenue of approach A; the second with B; etc.
7.2
4.1
The enemy combat power indices you read in are as follows:
7.20, 4.10.

How many friendly units are available for deployment? 3

Enter 3 friendly combat power indices. Separate each
 with a space or a <return>. Each one should correspond
 to a different friendly unit.
3
2.3
2
The friendly combat power indices you read in are as follows:
3.00, 2.30, 2.00.

 A near-optimal alignment of friendly units is as follows:

 Assign combat power index 3.00 to avenue of approach A.
 Assign combat power index 2.30 to avenue of approach B.
 Assign combat power index 2.00 to avenue of approach A.

 The Force Ratios are:

 A:  1.44;     B:  1.78;

-------------------------------------------
*    The near-optimal solution is 3.22.   *
-------------------------------------------
```

```
C:\QC2>aofa4
   ********************************************************
            This software aligns tactical forces
                       in the defense.

            It can assign up to 20 friendly units
             across 20 enemy avenues of approach.
   ********************************************************

How many enemy avenues of approach are known?  (Please enter
 an integer between 1 and 20) 3

Enter 3 enemy combat power indices. Separate each index
 with a space or a <return>. The first one should correspond
 with avenue of approach A; the second with B; etc.
14.1
8.6
10.7
The enemy combat power indices you read in are as follows:
14.10, 8.60, 10.70.

How many friendly units are available for deployment? 5

Enter 5 friendly combat power indices. Separate each
 with a space or a <return>. Each one should correspond
 to a different friendly unit.
3  2.1  1.7  3.5  2.3
The friendly combat power indices you read in are as follows:
3.00, 2.10, 1.70, 3.50, 2.30.

 A near-optimal alignment of friendly units is as follows:

 Assign combat power index 3.50 to avenue of approach A.
 Assign combat power index 3.00 to avenue of approach C.
 Assign combat power index 2.30 to avenue of approach B.
 Assign combat power index 2.10 to avenue of approach B.
 Assign combat power index 1.70 to avenue of approach A.

 The Force Ratios are:

 A:  2.71;    B:  1.95;    C:  3.57;

-----------------------------------------
*    The near-optimal solution is 8.23.   *
-----------------------------------------
```

```
A:\C>
   ************************************************************
           This software aligns tactical forces
                      in the defense.

           It can assign up to 20 friendly units
            across 20 enemy avenues of approach.
   ************************************************************

How many enemy avenues of approach are known?  (Please enter
 an integer between 1 and 20) 2

Enter 2 enemy combat power indices. Separate each index
 with a space or a <return>. The first one should correspond
 with avenue of approach A; the second with B; etc.
6.7 1.5
The enemy combat power indices you read in are as follows:
6.70, 1.50.

How many friendly units are available for deployment? 4

Enter 4 friendly combat power indices. Separate each
 with a space or a <return>. Each one should correspond
 to a different friendly unit.
2.7 1.3 1.4 1.9
The friendly combat power indices you read in are as follows:
2.70, 1.30, 1.40, 1.90.

 A near-optimal alignment of friendly units is as follows:

 Assign combat power index 2.70 to avenue of approach A.
 Assign combat power index 1.90 to avenue of approach A.
 Assign combat power index 1.40 to avenue of approach B.
 Assign combat power index 1.30 to avenue of approach B.

 The Force Ratios are:

 A:  1.46;     B:  0.56;

-------------------------------------------
*    The near-optimal solution is 2.01.    *
-------------------------------------------
```

```
C:\QC2>aofa4
   ************************************************************
           This software aligns tactical forces
                      in the defense.

           It can assign up to 20 friendly units
            across 20 enemy avenues of approach.
   ************************************************************

How many enemy avenues of approach are known?  (Please enter
 an integer between 1 and 20) 3

Enter 3 enemy combat power indices. Separate each index
 with a space or a <return>. The first one should correspond
 with avenue of approach A; the second with B; etc.
7.2
4.1
11.3
The enemy combat power indices you read in are as follows:
7.20, 4.10, 11.30.

How many friendly units are available for deployment? 4

Enter 4 friendly combat power indices. Separate each
 with a space or a <return>. Each one should correspond
 to a different friendly unit.
3  2.3  2  2.7
The friendly combat power indices you read in are as follows:
3.00, 2.30, 2.00, 2.70.

 A near-optimal alignment of friendly units is as follows:

 Assign combat power index 3.00 to avenue of approach C.
 Assign combat power index 2.70 to avenue of approach A.
 Assign combat power index 2.30 to avenue of approach B.
 Assign combat power index 2.00 to avenue of approach C.

 The Force Ratios are:

 A:  2.67;    B:  1.78;    C:  2.26;

 ----------------------------------------
 *    The near-optimal solution is 6.71.   *
 ----------------------------------------
```

Appendix D

ALGORITHM ANALYSIS

Table D-1 illustrates the effectiveness of the Near-Minimal Algorithm. The following notes apply to Table D-1:

ECPI is the Enemy Combat Power Index for each avenue of approach. The Wichman and Hill psuedo-random number generator at appendix E was used to generate numbers in the range 1 to 15. ECPIs will usually fall in this range.

FCPI is the Friendly Combat Power Index for each friendly unit. Again, the Wichman and Hill psuedo-random number generator at appendix E was used to generate FCPIs. The range of values were 1.5 to 3.5. FCPIs will usually fall in this range.

MIN is the minimal-feasible fractional integer solution to the sum of the force ratio objective.

MAX is the maximum-feasible fractional integer solution to the sum of the force ratio objective.

ALG'M is the near-minimal, feasible, algorithmic solution to the fractional integer force ratio objective.

% OPT is one minus the ratio of the absolute differences between ALG'M and MIN, and between MIN and MAX.

Appendix E

COMPUTER PROGRAMS

```
/* ---------------------------------------------------------------------
   This program applies the near-optimal force alignment algorithm
   in the defense developed by Mark E. Tillman. It can analyze up to
   20 enemy avenues of approach, and assign up to 20 friendly units.
   The author of this program is Mark E. Tillman, Colorado School of Mines.
                                   October, 1990
   The Indexx.c function was taken from "Numerical Recipes in C" and
   adapted to this program. The Sort.c function was taken from the
   Waite group's "Microsoft Quick-C Programming."
   I acknowledge the conceptual help of James Watson and Doug Hart. They
   each provided valuable hints and help on pointer operations.
   ---------------------------------------------------------------------
*/
#include <stdio.h>
#include <malloc.h>
#define FIX 20   /* Problem size definition and limitation. */
void Sort(float vals[], int flot);
void Indexx(int n, float arrin[], int indx[]);
main()
{

/* Declaration of variable types */
int c, i, j, s, index, size, flot;
int t[FIX];
int *int_vector(int, int);
int *ind;
int status=1;
float ecpi[FIX], fcpi[FIX], delta[FIX], denom[FIX];
float min=0;
printf("  ********************************************************\n");
printf("         This software aligns tactical forces\n");
printf("                 in the defense.\n\n");
printf("         It can assign up to %d friendly units\n", FIX);
printf("          across %d enemy avenues of approach.\n", FIX);
printf("  ********************************************************\n");
printf("\n\n");
```

```
/* Initialize arrays denom and t with zero values */
for(i=1; i<FIX+1; i++)
  {
  denom[i]=0;
  t[i]=0;
  }

/* Establish from user the # of avenues of approach */
 printf("How many enemy avenues of approach are known? (Please enter\n");
 printf(" an integer between 1 and %d) ____\b\b\b", FIX);
 scanf("%d", &size);

/* The tactical problem must have more than 1 avenue of approach */
 while(size == 1)
  {
  printf("\n************************ Error ************************");
  printf("\n Please reenter your number. It must exceed 1.____\b\b\b");
  scanf("%d", &size); /* User modifies input if error detected */
  }

/* User may not specify more avenues than problem size limitation */
 while (size > FIX)
  {
  printf("\n************************ Error ************************");
  printf("\n Please reenter your number. It should not exceed %d.", FIX);
  printf(" ____\b\b\b");
  scanf("%d", &size); /* User modifies input if error detected */
  }
 printf("\nEnter %d enemy combat power indices. Separate each index",size);
 printf("\n with a space or a <return>. The first one should correspond");
```

```
printf("\n with avenue of approach A; the second with B; etc.\n");
for(index=0; index < size; index++)
  scanf("%f", &ecpi[index]); /* User inputs indices for each avenue */
printf("The enemy combat power indices you read in are as follows:\n");
for(index=0; index<size-1; index++)
  printf("%.2f, ", ecpi[index]);
printf("%.2f.\n\n", ecpi[size-1]); /* System reads back input from user */

/* Dynamic allocation of array ind to # of avenues specified by user */
ind = int_vector(1, size);

/* Initialize array ind with consecutive positive integers */
for(i=1; i<size; i++)
  ind[i]=i;

/* Establish the # of friendly units from user */
printf("How many friendly units are available for deployment?");
printf(" ____\b\b\b");
scanf("%d", &flot);

/* Tactically, user must specify at least as many units as avenues */
while(flot < size)
  {
  printf("\n********************** Error ****************************");
  printf("\nYou must enter at least as many friendly units as avenues\n");
  printf("of approach. You have entered data for %d avenues", size);
  printf(" of approach.\nPlease reenter the number of friendly units");
  printf(" available for\ndeployment.____\b\b\b");
  scanf("%d", &flot); /* User modifies input if error detected */
  }
```

```
/* User enters indices for each unit */
 printf("\nEnter %d friendly combat power indices. Separate each\n", flot);
 printf(" with a space or a <return>. Each one should correspond\n");
 printf(" to a different friendly unit.\n");
 for(index=0; index < flot; index++)
   scanf("%f", &fcpi[index]);

/* System reads back input from user */
 printf("The friendly combat power indices you read in are as follows:\n");
 for(index=0; index<flot-1; index++)
   printf("%.2f, ", fcpi[index]);
 printf("%.2f.\n\n", fcpi[flot-1]);

/* Assign values to counter variables c and s. */
 c = flot - size;
 s = 0;

/* Sort array of friendly indices from high to low, altering the order */
 Sort(fcpi, flot);
 printf(" A near-optimal alignment of friendly units is as follows:\n\n");

/* Start algorithmic calculation with row 1 of worksheet */
 for(j=flot-1; j>=0; --j)
   {
   for(i=1; i<size+1; i++)  /* Calculate delta for each avenue */
    {
    if(denom[i] == 0) /* If no units are assigned to avenue i, */
     delta[i] = ecpi[i-1] - ecpi[i-1]/fcpi[j]; /* then delta equals this. */
    if(denom[i] != 0) /* If a unit is assigned to avenue i, */
     {
     if(s < c)      /* and feasibility is maintained, */
```

```
      /* then use this formula to calculate the delta. */
      delta[i] = ecpi[i-1]/denom[i] - ecpi[i-1]/(denom[i] + fcpi[j]);
     if(s >= c) /* If feasibility will be lost, */
         delta[i] = -10000.0; /* then assign delta this value. */
     }
 /* printf("The %dth delta is %.2f\n", i, delta[i]); */
   }

/* Sort all delta indices and preserve order */
 Indexx(size, delta, ind);
/* For the avenue with the greatest delta increment its position in array t */
 t[ind[size]]=t[ind[size]] + 1;

/* If this position is now greater than 1, increment the counter s */
 if(t[ind[size]] > 1)
   s++;
/* Assign new value to denom position with greastest delta */
 denom[ind[size]] += fcpi[j];

/* Change avenue of approach # to character output and print assignment */
 printf(" Assign combat power index %.2f", fcpi[j]);
 printf(" to avenue of approach %c.\n", ind[size]+64);

  /* printf("The biggest delta is %d\n", ind[size]); */
   }

/* Calculate and print all force ratios */
 printf("\n The Force Ratios are:\n\n");
 for(i=1; i<size+1; i++)
   {
```

```
   if(denom[i] != 0) /* Prohibits division by zero, should that occur */
    {
    printf(" %c:  %.2f;   ", i+64, ecpi[i-1]/denom[i]);

/* Calculate and print near optimal sum */
    min += ecpi[i-1]/denom[i];
    }
   }
 printf("\n\n----------------------------------------\n");
 printf("*    The near-optimal solution is %.2f.   *\n", min);
 printf("----------------------------------------\n\n");
}
```

```
/* This function sorts an array of floats from high to low, order altered */

void Sort(float vals[], int flot)
{
 int i, j;
 float temp;
 for(i=flot-1; i>0; --i)
   {
   for(j=0; j<i; ++j)
     {
     if(vals[j]>vals[j+1])
         {
         temp     = vals[j];
         vals[j]  = vals[j+1];
         vals[j+1] = temp;
         }
     }
   }
}

/* This function sorts the indexing of an array, preserving order. */

void Indexx(int n, float arrin[], int indx[])
{
 int l, j, ir, indxt, i;
 float q;
 for(j=1; j<=n; j++)
   indx[j]=j;
l = (n >> 1) + 1;
ir = n;
```

```
for(;;)
  {
  if(l > 1)
   q=arrin[(indxt=indx[--l])];
  else
   {
   q=arrin[(indxt=indx[ir])];
   indx[ir]=indx[1];
   if(--ir == 1)
    {
    indx[1]=indxt;
    return;
    }
   }
   i = l;
   j = l << 1;
   while(j <= ir)
         {
         if(j < ir && arrin[indx[j]] < arrin[indx[j+1]])
         j++;
         if(q < arrin[indx[j]])
          {
          indx[i]=indx[j];
          j += (i=j);
          }
         else
         j = ir+1;
         }
   indx[i]=indxt;
   }
}
```

```
/* This function allocates memory at run time (dynamically) */

int *int_vector(int low, int high)
{
 int *vector;
 vector = (int*)malloc((high-low+1)*sizeof(int));
 if (!vector)
 {
 printf("Error occurred in dynamic memory allocation.\n");
 exit(0);
 }
vector -= low;
return(vector);
}
```

```
/*-------------------------------------------------------------------

 This program is the random number generator proposed by Wichman and Hill.

 The author of this program is Mark E. Tillman, Colorado School of Mines.

                              October, 1990

 -------------------------------------------------------------------

*/

#include <stdio.h>

#include <math.h>

#define MOD_1 30269

#define MOD_2 30307

#define MOD_3 30323

main()

{

long int r[3];

void Rand(int n, long int r[]);

int i, n;

printf("Enter 3 different positive integers. These will be the seeds\n");

printf("for the Wichman and Hill random number generator.\n");

for(i=0; i<3; i++)

 scanf("%d", &r[i]);

printf("How many random numbers do you want to generate?\n");

 scanf("%d", &n);
```

```
for(i=0; i<n; i++)
  Rand (i, r);
}
void Rand(int n, long int r[])
{
float num, frac;
 r[0] = (171 * r[0]) % MOD_1;
 r[1] = (172 * r[1]) % MOD_2;
 r[2] = (170 * r[2]) % MOD_3;

 num = (r[0]/30269.0) + (r[1]/30307.0) + (r[2]/30323.0);
 frac = num - floor(num);
printf("Random number %d is %.1f\n", n+1, frac);
}
```

```
/* This program explicitly enumerates all possible solutions for 5 different
   sizes of fractional integer programming problems. Once enumerated the
   program finds the minimum and maximum objective function value. The 5
   different problems are:
   2 avenues of approach, 3 friendly units,
   2 avenues of approach, 4 friendly units,
   3 avenues of approach, 3 friendly units,
   3 avenues of approach, 4 friendly units,
   3 avenues of approach, 5 friendly units.
*/
#include <stdio.h>
#include <function.h>
void Sort(float vals[], int c);
void F23(float f[], float e[]);
void F24(float f[], float e[]);
void F33(float f[], float e[]);
void F34(float f[], float e[]);
void F35(float f[], float e[]);
main()
{
 int index, size, flot;
 float ecpi[7], fcpi[7], max[7];
 float x;
 printf("How many enemy avenues of approach are known?");
 printf(" (Please enter an\n integer between 1 and 7.)\n");
 scanf("%d", &size);
 while (size > 7)
  {
  printf("Please reenter your number. Be sure it is no more than 7.\n");
  scanf("%d", &size);
  }
 printf("Enter %d enemy combat power indices. The first one should\n", size);
 printf(" correspond with avenue of approach A; the second with B; etc.\n");
 for (index=0; index < size; index++)
  scanf("%f", &ecpi[index]);
```

```
printf("The enemy combat power indices you read in are as follows:\n");
for (index=0; index<size-1; index++)
 printf("%.2f, ", ecpi[index]);
printf("%.2f.\n\n", ecpi[size-1]);
printf("How many friendly units are available for deployment?\n");
scanf("%d", &flot);
printf("Enter %d friendly combat power indices. Each one should\n", flot);
printf(" correspond to a different friendly unit.\n");
for (index=0; index < flot; index ++)
 scanf("%f", &fcpi[index]);
printf("The friendly combat power indices you read in are as follows:\n");
for (index=0; index<flot-1; index++)
 printf("%.2f, ", fcpi[index]);
 printf("%.2f.\n\n", fcpi[flot-1]);
if(size==2)
 if(flot==3)
  F23(fcpi, ecpi);
if(size==2)
 if(flot==4)
  F24(fcpi, ecpi);
if(size==3)
 if(flot==3)
  F33(fcpi, ecpi);
if(size==3)
 if(flot==4)
  F34(fcpi, ecpi);
if(size==3)
 if(flot==5)
  {
 printf("My method will take me about 15 seconds. Please stand by...\n\n");
 F35(fcpi, ecpi);
  }
```

```
 if(size>3)
  if(flot>5)
   printf("Working on that problem...\n");
 if(size<2)
  printf("Please relook the problem. I don't see one here.\n");
 if(size==flot)
  if(size > 3)
   if(flot > 3)
        {
printf("This is a straight-laced assignment problem.  I am still working");
printf("on \nthis procedure.  It is not as difficult to solve, though.\n");
        }
}
```

```
/* This function sorts an array from low to high values. */
void Sort(float vals[], int c)
{
int i, j;
float temp;
for(i=c; i>=0; --i)
  {
  for(j=0; j<i; ++j)
    {
    if(vals[j]>vals[j+1])
          {
          temp     = vals[j];
          vals[j]  = vals[j+1];
          vals[j+1] = temp;
          }
    }
}
/* This function sums an array of floats. */
float Sum(float add[])
{
 int i;
 float total = 0;
 for(i = 0; i<6; i++)
   {
   total += *add;
   add++;
   }
 return total;
}
/* This function permutes the soln matrix for 3 units along 2 A's of A. */
```

```
void F23(float f[], float e[])
{
float a[3], s[6];
int c;
a[0]=f[0]+f[1]; a[1]=f[0]+f[2]; a[2]=f[1]+f[2];
s[0]=e[0]/a[0]+e[1]/f[2]; s[1]=e[1]/a[0]+e[0]/f[2]; s[2]=e[0]/a[1]+e[1]/f[1];
s[3]=e[1]/a[1]+e[0]/f[1]; s[4]=e[0]/a[2]+e[1]/f[0]; s[5]=e[1]/a[2]+e[0]/f[0];
c=5;
Sort(s, c);
printf("Enumeration finds the min soln to be %.2f.\n", s[0]);
printf("Enumeration finds the max soln to be %.2f.\n", s[5]);
}
/* This function permutes the soln matrix for 4 units along 2 A's of A. */
void F24(float f[], float e[])
{
float a[10], s[14];
int c;
a[0]=f[0]+f[1]+f[2]; a[1]=f[0]+f[1]+f[3]; a[2]=f[1]+f[2]+f[3];
a[3]=f[0]+f[2]+f[3]; a[4]=f[0]+f[1]; a[5]=f[2]+f[3]; a[6]=f[0]+f[3];
a[7]=f[1]+f[2]; a[8]=f[0]+f[3]; a[9]=f[2]+f[3];
s[0]=e[0]/a[0]+e[1]/f[3]; s[1]=e[1]/a[0]+e[2]/f[3]; s[2]=e[0]/a[1]+e[1]/f[2];
s[3]=e[1]/a[1]+e[0]/f[2]; s[4]=e[0]/a[2]+e[1]/f[0]; s[5]=e[1]/a[2]+e[2]/f[0];
s[6]=e[0]/a[3]+e[1]/f[1]; s[7]=e[1]/a[3]+e[0]/f[1];
s[8]=e[0]/a[4]+e[1]/a[5]; s[9]=e[1]/a[4]+e[0]/a[5]; s[10]=e[0]/a[6]+e[1]/a[7];
s[11]=e[1]/a[7]+e[0]/a[6];s[12]=e[0]/a[8]+e[1]/a[9];s[13]=e[1]/a[8]+e[0]/a[9];
c=13;
Sort(s, c);
printf("Enumeration finds the min soln to be %.2f.\n", s[0]);
printf("Enumeration finds the max soln to be %.2f.\n", s[13]);
}
/* This function permutes the soln matrix for 3 units along 3 A's of A. */
```

```
void F33(float f[], float e[])
{
float s[6];
int c;
s[0]=e[0]/f[0]+e[1]/f[1]+e[2]/f[2]; s[1]=e[0]/f[0]+e[1]/f[2]+e[2]/f[1];
s[2]=e[0]/f[1]+e[1]/f[0]+e[2]/f[2]; s[3]=e[0]/f[1]+e[1]/f[2]+e[2]/f[0];
s[4]=e[0]/f[2]+e[1]/f[1]+e[2]/f[0]; s[5]=e[0]/f[2]+e[1]/f[0]+e[2]/f[1];
c=5;
Sort(s, c);
printf("Enumeration finds the min soln to be %.2f.\n", s[0]);
printf("Enumeration finds the max soln to be %.2f.\n", s[5]);
}
/* This function permutes the soln matrix for 4 units along 3 A's of A. */
void F34(float f[], float e[])
{
float a[6], s[36];
int c;
a[0]=f[0]+f[1]; a[1]=f[2]+f[3]; a[2]=f[0]+f[2]; a[3]=f[1]+f[3]; a[4]=f[1]+f[2];
a[5]=f[0]+f[3];
s[0]=e[0]/a[0]+e[1]/f[2]+e[2]/f[3]; s[1]=e[0]/a[0]+e[1]/f[3]+e[2]/f[2];
s[2]=e[0]/f[2]+e[1]/a[0]+e[2]/f[3]; s[3]=e[0]/f[3]+e[1]/a[0]+e[2]/f[2];
s[4]=e[0]/f[3]+e[1]/f[2]+e[2]/a[0]; s[5]=e[0]/f[2]+e[1]/f[3]+e[2]/a[0];
s[6]=e[0]/a[1]+e[1]/f[0]+e[2]/f[1]; s[7]=e[0]/a[1]+e[1]/f[1]+e[2]/f[0];
s[8]=e[0]/f[0]+e[1]/a[1]+e[2]/f[1]; s[9]=e[0]/f[1]+e[1]/a[1]+e[2]/f[0];
s[10]=e[0]/f[0]+e[1]/f[1]+e[2]/a[1];s[11]=e[0]/f[1]+e[1]/f[0]+e[2]/a[1];
s[12]=e[0]/a[2]+e[1]/f[1]+e[2]/f[3];s[13]=e[0]/a[2]+e[1]/f[3]+e[2]/f[1];
s[14]=e[0]/f[1]+e[1]/a[2]+e[2]/f[3];s[15]=e[0]/f[3]+e[1]/a[2]+e[2]/f[1];
s[16]=e[0]/f[3]+e[1]/f[1]+e[2]/a[2];s[17]=e[0]/f[1]+e[1]/f[3]+e[2]/a[2];
s[18]=e[0]/a[3]+e[1]/f[2]+e[2]/f[0];s[19]=e[0]/a[3]+e[1]/f[0]+e[2]/f[2];
s[20]=e[0]/f[0]+e[1]/a[3]+e[2]/f[2];s[21]=e[0]/f[2]+e[1]/a[3]+e[2]/f[0];
s[22]=e[0]/f[0]+e[1]/f[2]+e[2]/a[3];s[23]=e[0]/f[2]+e[1]/f[0]+e[2]/a[3];
```

```
s[24]=e[0]/a[4]+e[1]/f[0]+e[2]/f[3];s[25]=e[0]/a[4]+e[1]/f[3]+e[2]/f[0];
s[26]=e[0]/f[0]+e[1]/a[4]+e[2]/f[3];s[27]=e[0]/f[3]+e[1]/a[4]+e[2]/f[0];
s[28]=e[0]/f[3]+e[1]/f[0]+e[2]/a[4];s[29]=e[0]/f[0]+e[1]/f[3]+e[2]/a[4];
s[30]=e[0]/a[5]+e[1]/f[2]+e[2]/f[1];s[31]=e[0]/a[5]+e[1]/f[1]+e[2]/f[2];
s[32]=e[0]/f[2]+e[1]/a[5]+e[2]/f[1];s[33]=e[0]/f[1]+e[1]/a[5]+e[2]/f[2];
s[34]=e[0]/f[1]+e[1]/f[2]+e[2]/a[5];s[35]=e[0]/f[2]+e[1]/f[1]+e[2]/a[5];
c=35;
Sort(s, c);
printf("Enumeration finds the min soln to be %.2f.\n", s[0]);
printf("Enumeration finds the max soln to be %.2f.\n", s[35]);
}
/* This function permutes the soln matrix for 5 units along 3 A's of A. */
void F35(float f[], float e[])
{
float a[20], s[240];
int c;
a[0]=f[0]+f[1]; a[1]=f[0]+f[2]; a[2]=f[0]+f[3]; a[3]=f[0]+f[4]; a[4]=f[1]+f[2];
a[5]=f[1]+f[3]; a[6]=f[1]+f[4]; a[7]=f[2]+f[4]; a[8]=f[2]+f[3]; a[9]=f[3]+f[4];
a[10]=f[0]+f[1]+f[2]; a[11]=f[0]+f[1]+f[3]; a[12]=f[0]+f[1]+f[4];
a[13]=f[0]+f[2]+f[3]; a[14]=f[0]+f[2]+f[4]; a[15]=f[0]+f[3]+f[4];
a[16]=f[1]+f[2]+f[3]; a[17]=f[1]+f[2]+f[4]; a[18]=f[1]+f[3]+f[4];
a[19]=f[2]+f[3]+f[4];
s[0]=e[0]/a[0]+e[1]/a[8]+e[2]/f[4]; s[1]=e[0]/a[0]+e[2]/a[8]+e[1]/f[4];
s[2]=e[1]/a[0]+e[2]/a[8]+e[0]/f[4]; s[3]=e[1]/a[0]+e[0]/a[8]+e[2]/f[4];
s[4]=e[2]/a[0]+e[1]/a[8]+e[0]/f[4]; s[5]=e[2]/a[0]+e[0]/a[8]+e[1]/f[4];
s[6]=e[0]/a[0]+e[1]/a[7]+e[2]/f[3]; s[7]=e[0]/a[0]+e[2]/a[7]+e[1]/f[3];
s[8]=e[1]/a[0]+e[2]/a[7]+e[0]/f[3]; s[9]=e[1]/a[0]+e[0]/a[7]+e[2]/f[3];
s[10]=e[2]/a[0]+e[1]/a[7]+e[0]/f[3];s[11]=e[2]/a[0]+e[0]/a[7]+e[1]/f[3];
s[12]=e[0]/a[0]+e[1]/a[9]+e[2]/f[2];s[13]=e[0]/a[0]+e[2]/a[9]+e[1]/f[2];
s[14]=e[1]/a[0]+e[2]/a[9]+e[0]/f[2];s[15]=e[1]/a[0]+e[0]/a[9]+e[2]/f[2];
s[16]=e[2]/a[0]+e[1]/a[9]+e[0]/f[2];s[17]=e[2]/a[0]+e[0]/a[9]+e[1]/f[2];
```

```
s[18]=e[0]/a[1]+e[1]/a[5]+e[2]/f[4];s[19]=e[0]/a[1]+e[2]/a[5]+e[1]/f[4];
s[20]=e[1]/a[1]+e[2]/a[5]+e[0]/f[4];s[21]=e[1]/a[1]+e[0]/a[5]+e[2]/f[4];
s[22]=e[2]/a[1]+e[1]/a[5]+e[0]/f[4];s[23]=e[2]/a[1]+e[0]/a[5]+e[1]/f[4];
s[24]=e[0]/a[1]+e[1]/a[6]+e[2]/f[3];s[25]=e[0]/a[1]+e[2]/a[6]+e[1]/f[3];
s[26]=e[1]/a[1]+e[2]/a[6]+e[0]/f[3];s[27]=e[1]/a[1]+e[0]/a[6]+e[2]/f[3];
s[28]=e[2]/a[1]+e[1]/a[6]+e[0]/f[3];s[29]=e[2]/a[1]+e[0]/a[6]+e[1]/f[3];
s[30]=e[0]/a[1]+e[1]/a[9]+e[2]/f[1];s[31]=e[0]/a[1]+e[2]/a[9]+e[1]/f[1];
s[32]=e[1]/a[1]+e[2]/a[9]+e[0]/f[1];s[33]=e[1]/a[1]+e[0]/a[9]+e[2]/f[1];
s[34]=e[2]/a[1]+e[1]/a[9]+e[0]/f[1];s[35]=e[2]/a[1]+e[0]/a[9]+e[1]/f[1];
s[36]=e[0]/a[2]+e[1]/a[7]+e[2]/f[1];s[37]=e[0]/a[2]+e[2]/a[7]+e[1]/f[1];
s[38]=e[1]/a[2]+e[2]/a[7]+e[0]/f[1];s[39]=e[1]/a[2]+e[0]/a[7]+e[2]/f[1];
s[40]=e[2]/a[2]+e[1]/a[7]+e[0]/f[1];s[41]=e[2]/a[2]+e[0]/a[7]+e[1]/f[1];
s[42]=e[0]/a[2]+e[1]/a[4]+e[2]/f[4];s[43]=e[0]/a[2]+e[2]/a[4]+e[1]/f[4];
s[44]=e[1]/a[2]+e[2]/a[4]+e[0]/f[4];s[45]=e[1]/a[2]+e[0]/a[4]+e[2]/f[4];
s[46]=e[2]/a[2]+e[1]/a[4]+e[0]/f[4];s[47]=e[2]/a[2]+e[0]/a[4]+e[1]/f[4];
s[48]=e[0]/a[2]+e[1]/a[6]+e[2]/f[2];s[49]=e[0]/a[2]+e[2]/a[6]+e[1]/f[2];
s[50]=e[1]/a[2]+e[2]/a[6]+e[0]/f[2];s[51]=e[1]/a[2]+e[0]/a[6]+e[2]/f[2];
s[52]=e[2]/a[2]+e[1]/a[6]+e[0]/f[2];s[53]=e[2]/a[2]+e[0]/a[6]+e[1]/f[2];
s[54]=e[0]/a[3]+e[1]/a[4]+e[2]/f[3];s[55]=e[0]/a[3]+e[2]/a[4]+e[1]/f[3];
s[56]=e[1]/a[3]+e[2]/a[4]+e[0]/f[3];s[57]=e[1]/a[3]+e[0]/a[4]+e[2]/f[3];
s[58]=e[2]/a[3]+e[1]/a[4]+e[0]/f[3];s[59]=e[2]/a[3]+e[0]/a[4]+e[1]/f[3];
s[60]=e[0]/a[3]+e[1]/a[8]+e[2]/f[1];s[61]=e[0]/a[3]+e[2]/a[8]+e[1]/f[1];
s[62]=e[1]/a[3]+e[2]/a[8]+e[0]/f[1];s[63]=e[1]/a[3]+e[0]/a[8]+e[2]/f[1];
s[64]=e[2]/a[3]+e[1]/a[8]+e[0]/f[1];s[65]=e[2]/a[3]+e[0]/a[8]+e[1]/f[1];
s[66]=e[0]/a[3]+e[1]/a[5]+e[2]/f[2];s[67]=e[0]/a[3]+e[2]/a[5]+e[1]/f[2];
s[68]=e[1]/a[3]+e[2]/a[5]+e[0]/f[2];s[69]=e[1]/a[3]+e[0]/a[5]+e[2]/f[2];
s[70]=e[2]/a[3]+e[1]/a[5]+e[0]/f[2];s[71]=e[2]/a[3]+e[0]/a[5]+e[1]/f[2];
s[72]=e[0]/a[4]+e[1]/a[9]+e[2]/f[0];s[73]=e[0]/a[4]+e[2]/a[9]+e[1]/f[0];
s[74]=e[1]/a[4]+e[2]/a[9]+e[0]/f[0];s[75]=e[1]/a[4]+e[0]/a[9]+e[2]/f[0];
s[76]=e[2]/a[4]+e[1]/a[9]+e[0]/f[0];s[77]=e[2]/a[4]+e[0]/a[9]+e[1]/f[0];
```

s[78]=e[0]/a[4]+e[1]/a[2]+e[2]/f[4];s[79]=e[0]/a[4]+e[2]/a[2]+e[1]/f[4];
s[80]=e[1]/a[4]+e[2]/a[2]+e[0]/f[4];s[81]=e[1]/a[4]+e[0]/a[2]+e[2]/f[4];
s[82]=e[2]/a[4]+e[1]/a[2]+e[0]/f[4];s[83]=e[2]/a[4]+e[0]/a[2]+e[1]/f[4];
s[84]=e[0]/a[4]+e[1]/a[3]+e[2]/f[3];s[85]=e[0]/a[4]+e[2]/a[3]+e[1]/f[3];
s[86]=e[1]/a[4]+e[2]/a[3]+e[0]/f[3];s[87]=e[1]/a[4]+e[0]/a[3]+e[2]/f[3];
s[88]=e[2]/a[4]+e[1]/a[3]+e[0]/f[3];s[89]=e[2]/a[4]+e[0]/a[3]+e[1]/f[3];
s[90]=e[0]/a[5]+e[1]/a[1]+e[2]/f[4];s[91]=e[0]/a[5]+e[2]/a[1]+e[1]/f[4];
s[92]=e[1]/a[5]+e[2]/a[1]+e[0]/f[4];s[93]=e[1]/a[5]+e[0]/a[1]+e[2]/f[4];
s[94]=e[2]/a[5]+e[1]/a[1]+e[0]/f[4];s[95]=e[2]/a[5]+e[0]/a[1]+e[1]/f[4];
s[96]=e[0]/a[5]+e[1]/a[3]+e[2]/f[2];s[97]=e[0]/a[5]+e[2]/a[3]+e[1]/f[2];
s[98]=e[1]/a[5]+e[2]/a[3]+e[0]/f[2];s[99]=e[1]/a[5]+e[0]/a[3]+e[2]/f[2];
s[100]=e[2]/a[5]+e[1]/a[3]+e[0]/f[2];s[101]=e[2]/a[5]+e[0]/a[3]+e[1]/f[2];
s[102]=e[0]/a[5]+e[1]/a[7]+e[2]/f[0];s[103]=e[0]/a[5]+e[2]/a[7]+e[1]/f[0];
s[104]=e[1]/a[5]+e[2]/a[7]+e[0]/f[0];s[105]=e[1]/a[5]+e[0]/a[7]+e[2]/f[0];
s[106]=e[2]/a[5]+e[1]/a[7]+e[0]/f[0];s[107]=e[2]/a[5]+e[0]/a[7]+e[1]/f[0];
s[108]=e[0]/a[6]+e[1]/a[1]+e[2]/f[3];s[109]=e[0]/a[6]+e[2]/a[1]+e[1]/f[3];
s[110]=e[1]/a[6]+e[2]/a[1]+e[0]/f[3];s[111]=e[1]/a[6]+e[0]/a[1]+e[2]/f[3];
s[112]=e[2]/a[6]+e[1]/a[1]+e[0]/f[3];s[113]=e[2]/a[6]+e[0]/a[1]+e[1]/f[3];
s[114]=e[0]/a[6]+e[1]/a[8]+e[2]/f[0];s[115]=e[0]/a[6]+e[2]/a[8]+e[1]/f[0];
s[116]=e[1]/a[6]+e[2]/a[8]+e[0]/f[0];s[117]=e[1]/a[6]+e[0]/a[8]+e[2]/f[0];
s[118]=e[2]/a[6]+e[1]/a[8]+e[0]/f[0];s[119]=e[2]/a[6]+e[0]/a[8]+e[1]/f[0];
s[120]=e[0]/a[6]+e[1]/a[2]+e[2]/f[2];s[121]=e[0]/a[6]+e[2]/a[2]+e[1]/f[2];
s[122]=e[1]/a[6]+e[2]/a[2]+e[0]/f[2];s[123]=e[1]/a[6]+e[0]/a[2]+e[2]/f[2];
s[124]=e[2]/a[6]+e[1]/a[2]+e[0]/f[2];s[125]=e[2]/a[6]+e[0]/a[2]+e[1]/f[2];
s[126]=e[0]/a[7]+e[1]/a[0]+e[2]/f[3];s[127]=e[0]/a[7]+e[2]/a[0]+e[1]/f[3];
s[128]=e[1]/a[7]+e[2]/a[0]+e[0]/f[3];s[129]=e[1]/a[7]+e[0]/a[0]+e[2]/f[3];
s[130]=e[2]/a[7]+e[1]/a[0]+e[0]/f[3];s[131]=e[2]/a[7]+e[0]/a[0]+e[1]/f[3];
s[132]=e[0]/a[7]+e[1]/a[2]+e[2]/f[1];s[133]=e[0]/a[7]+e[2]/a[2]+e[1]/f[1];
s[134]=e[1]/a[7]+e[2]/a[2]+e[0]/f[1];s[135]=e[1]/a[7]+e[0]/a[2]+e[2]/f[1];
s[136]=e[2]/a[7]+e[1]/a[2]+e[0]/f[1];s[137]=e[2]/a[7]+e[0]/a[2]+e[1]/f[1];

s[138]=e[0]/a[7]+e[1]/a[5]+e[2]/f[0];s[139]=e[0]/a[7]+e[2]/a[5]+e[1]/f[0];
s[140]=e[1]/a[7]+e[2]/a[5]+e[0]/f[0];s[141]=e[1]/a[7]+e[0]/a[5]+e[2]/f[0];
s[142]=e[2]/a[7]+e[1]/a[5]+e[0]/f[0];s[143]=e[2]/a[7]+e[0]/a[5]+e[1]/f[0];
s[144]=e[0]/a[8]+e[1]/a[0]+e[2]/f[4];s[145]=e[0]/a[8]+e[2]/a[0]+e[1]/f[4];
s[146]=e[1]/a[8]+e[2]/a[0]+e[0]/f[4];s[147]=e[1]/a[8]+e[0]/a[0]+e[2]/f[4];
s[148]=e[2]/a[8]+e[1]/a[0]+e[0]/f[4];s[149]=e[2]/a[8]+e[0]/a[0]+e[1]/f[4];
s[150]=e[0]/a[8]+e[1]/a[4]+e[2]/f[1];s[151]=e[0]/a[8]+e[2]/a[4]+e[1]/f[1];
s[152]=e[1]/a[8]+e[2]/a[4]+e[0]/f[1];s[153]=e[1]/a[8]+e[0]/a[4]+e[2]/f[1];
s[154]=e[2]/a[8]+e[1]/a[4]+e[0]/f[1];s[155]=e[2]/a[8]+e[0]/a[4]+e[1]/f[1];
s[156]=e[0]/a[8]+e[1]/a[6]+e[2]/f[0];s[157]=e[0]/a[8]+e[2]/a[6]+e[1]/f[0];
s[158]=e[1]/a[8]+e[2]/a[6]+e[0]/f[0];s[159]=e[1]/a[8]+e[0]/a[6]+e[2]/f[0];
s[160]=e[2]/a[8]+e[1]/a[6]+e[0]/f[0];s[161]=e[2]/a[8]+e[0]/a[6]+e[1]/f[0];
s[162]=e[0]/a[9]+e[1]/a[0]+e[2]/f[2];s[163]=e[0]/a[9]+e[2]/a[0]+e[1]/f[2];
s[164]=e[1]/a[9]+e[2]/a[0]+e[0]/f[2];s[165]=e[1]/a[9]+e[0]/a[0]+e[2]/f[2];
s[166]=e[2]/a[9]+e[1]/a[0]+e[0]/f[2];s[167]=e[2]/a[9]+e[0]/a[0]+e[1]/f[2];
s[168]=e[0]/a[9]+e[1]/a[1]+e[2]/f[1];s[169]=e[0]/a[9]+e[2]/a[1]+e[1]/f[1];
s[170]=e[1]/a[9]+e[2]/a[1]+e[0]/f[1];s[171]=e[1]/a[9]+e[0]/a[1]+e[2]/f[1];
s[172]=e[2]/a[9]+e[1]/a[1]+e[0]/f[1];s[173]=e[2]/a[9]+e[0]/a[1]+e[1]/f[1];
s[174]=e[0]/a[9]+e[1]/a[4]+e[2]/f[0];s[175]=e[0]/a[9]+e[2]/a[4]+e[1]/f[0];
s[176]=e[1]/a[9]+e[2]/a[4]+e[0]/f[0];s[177]=e[1]/a[9]+e[0]/a[4]+e[2]/f[0];
s[178]=e[2]/a[9]+e[1]/a[4]+e[0]/f[0];s[179]=e[2]/a[9]+e[0]/a[4]+e[1]/f[0];
s[180]=e[0]/a[10]+e[1]/f[3]+e[2]/f[4];s[181]=e[0]/a[10]+e[2]/f[3]+e[1]/f[4];
s[182]=e[1]/a[10]+e[2]/f[3]+e[0]/f[4];s[183]=e[1]/a[10]+e[0]/f[3]+e[2]/f[4];
s[184]=e[2]/a[10]+e[1]/f[3]+e[0]/f[4];s[185]=e[2]/a[10]+e[0]/f[3]+e[1]/f[4];
s[186]=e[0]/a[11]+e[1]/f[2]+e[2]/f[4];s[187]=e[0]/a[11]+e[2]/f[2]+e[1]/f[4];
s[188]=e[1]/a[11]+e[2]/f[2]+e[0]/f[4];s[189]=e[1]/a[11]+e[0]/f[2]+e[2]/f[4];
s[190]=e[2]/a[11]+e[1]/f[2]+e[0]/f[4];s[191]=e[2]/a[11]+e[0]/f[2]+e[1]/f[4];
s[192]=e[0]/a[12]+e[1]/f[2]+e[2]/f[3];s[193]=e[0]/a[12]+e[2]/f[2]+e[1]/f[3];
s[194]=e[1]/a[12]+e[2]/f[2]+e[0]/f[3];s[195]=e[1]/a[12]+e[0]/f[2]+e[2]/f[3];
s[196]=e[2]/a[12]+e[1]/f[2]+e[0]/f[3];s[197]=e[2]/a[12]+e[0]/f[2]+e[1]/f[3];

```
s[198]=e[0]/a[13]+e[1]/f[1]+e[2]/f[4];s[199]=e[0]/a[13]+e[2]/f[1]+e[1]/f[4];
s[200]=e[1]/a[13]+e[2]/f[1]+e[0]/f[4];s[201]=e[1]/a[13]+e[0]/f[1]+e[2]/f[4];
s[202]=e[2]/a[13]+e[1]/f[1]+e[0]/f[4];s[203]=e[2]/a[13]+e[0]/f[1]+e[1]/f[4];
s[204]=e[0]/a[14]+e[1]/f[1]+e[2]/f[3];s[205]=e[0]/a[14]+e[2]/f[1]+e[1]/f[3];
s[206]=e[1]/a[14]+e[2]/f[1]+e[0]/f[3];s[207]=e[1]/a[14]+e[0]/f[1]+e[2]/f[3];
s[208]=e[2]/a[14]+e[1]/f[1]+e[0]/f[3];s[209]=e[2]/a[14]+e[0]/f[1]+e[1]/f[3];
s[210]=e[0]/a[15]+e[1]/f[1]+e[2]/f[2];s[211]=e[0]/a[15]+e[2]/f[1]+e[1]/f[2];
s[212]=e[1]/a[15]+e[2]/f[1]+e[0]/f[2];s[213]=e[1]/a[15]+e[0]/f[1]+e[2]/f[2];
s[214]=e[2]/a[15]+e[1]/f[1]+e[0]/f[2];s[215]=e[2]/a[15]+e[0]/f[1]+e[1]/f[2];
s[216]=e[0]/a[16]+e[1]/f[0]+e[2]/f[4];s[217]=e[0]/a[16]+e[2]/f[0]+e[1]/f[4];
s[218]=e[1]/a[16]+e[2]/f[0]+e[0]/f[4];s[219]=e[1]/a[16]+e[0]/f[0]+e[2]/f[4];
s[220]=e[2]/a[16]+e[1]/f[0]+e[0]/f[4];s[221]=e[2]/a[16]+e[0]/f[0]+e[1]/f[4];
s[222]=e[0]/a[17]+e[1]/f[0]+e[2]/f[3];s[223]=e[0]/a[17]+e[2]/f[0]+e[1]/f[3];
s[224]=e[1]/a[17]+e[2]/f[0]+e[0]/f[3];s[225]=e[1]/a[17]+e[0]/f[0]+e[2]/f[3];
s[226]=e[2]/a[17]+e[1]/f[0]+e[0]/f[3];s[227]=e[2]/a[17]+e[0]/f[0]+e[1]/f[3];
s[228]=e[0]/a[18]+e[1]/f[0]+e[2]/f[2];s[229]=e[0]/a[18]+e[2]/f[0]+e[1]/f[2];
s[230]=e[1]/a[18]+e[2]/f[0]+e[0]/f[2];s[231]=e[1]/a[18]+e[0]/f[0]+e[2]/f[2];
s[232]=e[2]/a[18]+e[1]/f[0]+e[0]/f[2];s[233]=e[2]/a[18]+e[0]/f[0]+e[1]/f[2];
s[234]=e[0]/a[19]+e[1]/f[0]+e[2]/f[1];s[235]=e[0]/a[19]+e[2]/f[0]+e[1]/f[1];
s[236]=e[1]/a[19]+e[2]/f[0]+e[0]/f[1];s[237]=e[1]/a[19]+e[0]/f[0]+e[2]/f[1];
s[238]=e[2]/a[19]+e[1]/f[0]+e[0]/f[1];s[239]=e[2]/a[19]+e[0]/f[0]+e[1]/f[1];
c=239;
Sort(s, c);
printf("Enumeration finds the min soln to be %.2f.\n", s[0]);
printf("Enumeration finds the max soln to be %.2f.\n", s[239]);
}
```